NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

TWO-WAY PATTERN DESIGN FOR DISTRIBUTED SUBARRAY ANTENNAS

by

Hock Hin Cher

September 2012

Thesis Advisor: David C. Jenn
Second Reader: Tri T. Ha

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2012	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE Two-Way Pattern Design for Distributed Subarray Antennas			**5. FUNDING NUMBERS**
6. AUTHOR(S) Hock Hin Cher			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number ______N/A______.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

Modern phased array radar uses multifunction subarray antennas in a distributed fashion. Distributed subarrays (DSA) have the advantages of more efficient scheduling of track and search functions, rapid steering capability, decreased complexity in digital beamforming and better angular resolution. However, one disadvantage of the DSA are the extra grating lobes due to large subarray spacing which can cause ambiguities in angle measurements and excess background clutter. A possible approach to suppress the grating lobes is to design separate transmit and receive subarray antennas that have different radiation patterns.

The purpose of this research was to develop a program based on the principle of pattern multiplication to synthesize and access the two-way antenna pattern for DSAs. The program, written in MATLAB, allows the user to study the two-way antenna pattern for different subarray architectures. The program was able to synthesize the pattern for isotropic elements, short dipoles and half-wave dipoles in a planar array above a ground plane. A simulation tool was also developed to map the grating lobe and null locations of the antenna patterns in direction cosine space. Several DSA configurations were examined, and the results showed that undesired grating lobes can be suppressed by subarray nulls.

14. SUBJECT TERMS Subarray, Two-way Pattern, Distributed Subarray Antenna, Phased Array, Pattern Multiplication			**15. NUMBER OF PAGES** 144
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

TWO-WAY PATTERN DESIGN FOR DISTRIBUTED SUBARRAY ANTENNAS

Hock Hin Cher
Civilian, ST Electronics, Singapore
B.S., Singapore Institute of Management, 2004

Submitted in partial fulfillment of the
requirements for the degree of

**MASTER OF SCIENCE IN ENGINEERING SCIENCE
(ELECTRICAL ENGINEERING)**

from the

**NAVAL POSTGRADUATE SCHOOL
September 2012**

Author: Hock Hin Cher

Approved by: David C. Jenn
Thesis Advisor

Tri T. Ha
Second Reader

R. Clark Robertson
Chair, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Modern phased array radar uses multifunction subarray antennas in a distributed fashion. Distributed subarrays (DSA) have the advantages of more efficient scheduling of track and search functions, rapid steering capability, decreased complexity in digital beamforming and better angular resolution. However, one disadvantage of the DSA are the extra grating lobes due to large subarray spacing which can cause ambiguities in angle measurements and excess background clutter. A possible approach to suppress the grating lobes is to design separate transmit and receive subarray antennas that have different radiation patterns.

The purpose of this research was to develop a program based on the principle of pattern multiplication to synthesize and access the two-way antenna pattern for DSAs. The program, written in MATLAB, allows the user to study the two-way antenna pattern for different subarray architectures. The program was able to synthesize the pattern for isotropic elements, short dipoles and half-wave dipoles in a planar array above a ground plane. A simulation tool was also developed to map the grating lobe and null locations of the antenna patterns in direction cosine space. Several DSA configurations were examined, and the results showed that undesired grating lobes can be suppressed by subarray nulls.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

AF	Array Factor
BWFN	Beamwidth between First Nulls
DAR	Distributed Array Radar
DSA	Distributed Subarray
EF	Element Factor
GF	Ground Plane Factor
GUI	Graphical User Interface
HPBW	Half-power Beamwidth
MFR	Multifunction Radar
RCS	Radar Cross Section
RRE	Radar Range Equation
SLL	Sidelobe Level

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

Modern phased array radar uses multifunction subarray antennas in a distributed fashion. Distributed subarrays (DSA) have the advantages of more efficient scheduling of track and search functions, rapid steering capability, decreased complexity in digital beamforming and better angular resolution. One disadvantage of the DSA are the extra grating lobes due to large subarray spacing which can cause ambiguities in angle measurements and excess background clutter. A possible approach to suppress the grating lobes is to design separate transmit and receive subarray antennas that have different radiation patterns.

The primary objective of this research was to develop a simulation tool to investigate the behavior and effectiveness in suppressing undesired grating lobes using the approach of two-way antenna pattern in DSA design. The fundamental array theory and principle of pattern multiplication, which formed the basis of the simulation tool design, were discussed. With the principle of pattern multiplication, grating lobes can be suppressed by placement of subarray nulls at grating lobe locations. A simple program was developed in MATLAB to allow the user to visualize the placement of grating lobes and nulls in direction cosine space for a DSA configuration in the visible region. A simulation tool with graphical user interface (GUI) was developed and implemented in MATLAB to perform the two-way antenna pattern and power gain calculations for user configured DSAs. The program is capable of performing two-way pattern and power gain calculations for linear or planar DSAs consisting of isotropic elements, half-wave dipoles or short dipoles above a ground plane. The program is able to present the simulation results in the θ pattern cut, ϕ pattern cut, or as a three-dimensional mesh plot in direction cosine space. The program GUI provides a convenient way for the user to tweak the design configurations very quickly by changing the DSA parameters.

The effectiveness of the two-way pattern multiplication approach to suppress undesired grating lobes by placement of subarray nulls at the grating lobe locations was demonstrated using the simulation tool. For a transmit DSA with subarray spacing of five wavelengths and uniform amplitude illumination, a sidelobe level (SLL) of −25.5 dB was achieved. Using the simulation tool, we demonstrated that low SLL and narrow half-power beamwidth (HPBW) of the two-way antenna pattern can be achieved using thinned transmit and receive arrays consisting of widely-spaced subarrays with non-coincident grating lobe locations. The simulation results showed that a peak SLL of −49.7 dB and HPBW of 0.9° were obtained for broadside illumination. Simulation was also carried out to examine the effects on the two-way pattern when different scan angles were applied. The simulation results showed that beam broadening was evident and peak SLL was increased for the two-way antenna pattern when large scan angles were applied to the main beams of the transmit and receive arrays. It was assumed that the main beams of the transmit and receive arrays were scanned to the same angle.

The simulation tool that was successfully developed for this research will serve as a useful tool for both students and electromagnetic professionals to determine and study the two-way pattern and power gain of different transmit/receive DSA designs. The simulation tool has also demonstrated the advantages of using separate transmit and receive antenna patterns to suppress undesired grating lobes and achieve narrow two-way beamwidth using fewer antenna elements.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my thesis advisor, Professor David C. Jenn for his patience, support and advice. Without his valuable knowledge and guidance, this thesis would not have been completed.

I would like to thank Professor Tri. T. Ha, for agreeing to be the second reader of this thesis.

I would like to thank my lovely wife, Justina Chua, for her unwavering support, understanding and patience, since I could not spend enough time with her while I was preparing for this research.

Lastly, I would like to thank Singapore Technologies (Electronics), and my superiors, Mr. Teai Yam Koon and Mr. Yeo Eng Choon, for providing me this opportunity to study for my master's degree at the Naval Postgraduate School.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND

A phased array is a directive antenna made up of a number of individual antennas, or radiating elements [1]. These individual antennas are geometrically arranged and excited by relatively phased currents to produce desired radiation patterns. Since the 1980s, phased arrays have gained wide use in military radar and communication applications. The Aegis ships in the US Navy use phased arrays for the AN/SPY-1 multifunction radar (MFR) [2]. The phased array for the AN/SPY-1 radar is shown in Figure 1.

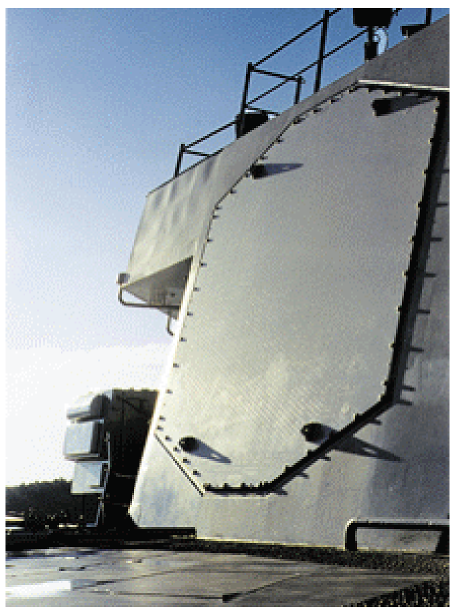


Figure 1. AN/SPY-1 phased array radar (From [3]).

With the advent of stealth technology to reduce the radar cross section (RCS) of modern naval fighting ships, platform real estate has become a precious commodity. This leaves limited areas on the ship structure for large antenna arrays. This is a challenge for design of antenna apertures for shipboard radar systems. One design approach is to use multi-function subarray antennas in a distributed fashion. A distributed subarray (DSA) consists of multiple subarrays of antenna elements physically distributed over a platform to form a long baseline, very thinned array for accurate angular location of targets [4]. Synthetic apertures can be generated using coherent processing to produce optimized patterns for specific radar applications. DSAs have the advantages of more efficient scheduling of track and search functions, rapid steering capability, decreased complexity in digital beamforming, and better angular resolution. One disadvantage of the DSA is the extra grating lobes that are introduced due to large subarray spacing of multiple wavelengths. Grating lobes, a form of undesired aliasing, exist when more than one period of the array factor appears in the visible region. Grating lobes are undesirable because they have the same magnitude as the main beam, which can cause ambiguities in angle measurements and excess background clutter.

A possible approach to suppress the grating lobes is to design separate transmit and receive subarray antennas. In radar systems, the received power depends on the product of transmit and receive antenna gains. In theory, the undesired grating lobes can be suppressed by combining transmit and receive antenna pairs that have different radiation patterns. The separate receive antenna can be designed with nulls in the direction of the transmit antenna array’s grating lobes in order to cancel or reduce its effect by taking advantage of the two-way antenna gain in the radar range equation.

The major advantages of using separate transmit and receive antenna patterns are the following:

- A narrow two-way beamwidth can be achieved using fewer antenna elements.
- An increase in transmit/receive isolation can be achieved.
- There is more flexibility in the physical placement of the antennas.
- There is an added dimension of beamforming for digital phased arrays.
- There is a possibility of clutter reduction using pattern design.

B. PREVIOUS RESEARCH

The concept of distributed array radar (DAR) and factors involving two-way effective gain patterns for DAR arrays are discussed in [4]. The study described a convolution approach to evaluate DAR resolution patterns using different transmit and receive array apertures. Lin [5] described methods of improving angular resolution of shipboard radar using DSAs. His research also proposed using two-way patterns to suppress grating lobes in widely spaced subarrays.

C. SCOPE OF RESEARCH

The objective of this research is to design a simulation tool to investigate the behavior and effectiveness in suppressing undesired grating lobes by using the approach of two-way antenna pattern in DSA design. The element factor for short dipoles and half-wave dipoles above a ground plane is included. The array factor and principle of pattern multiplication are employed to reduce the computational burden. A program is developed using MATLAB to synthesize and evaluate the two-way antenna pattern for planar phased arrays. Several DSA configurations are examined, and the synthesis results presented and discussed.

D. ORGANIZATION OF THESIS

The overall thesis report consists of five chapters. The scope of the thesis and background, which includes a brief introduction of distributed phased arrays, is covered in Chapter I.

The fundamental theory of phased arrays is discussed in Chapter II. The scanning pattern for arrays with elements above a ground plane is presented in this chapter. The array factor, ground plane factor, and element factor are introduced. In addition, the effects of grating lobes due to periodic variations in the array are explained.

The principles of digital array beamforming, directivity, gain, and radar range equations for DSAs are introduced in Chapter III. The approach to suppress undesired grating lobes using the principle of pattern multiplication is also discussed.

The program developed and implemented in MATLAB for simulating the two-way antenna pattern for DSAs is described in Chapter IV. The simulation results conducted for several DSA configurations are presented and analyzed, as well.

Finally, the summary and recommendations for future research and study are provided in Chapter V.

The MATLAB codes for the program developed for this research are provided in the Appendix.

II. ARRAY PATTERN THEORY

In this chapter, the fundamental theory of array antennas for a two-dimensional array above a ground plane is presented. The array pattern, array factor, element factor, and ground plane factor are introduced. Grating lobes in DSA design and their effects are also described.

A. ARRAY ANTENNAS

An array can be considered as a collection of identical radiating elements that are excited to achieve some desired radiating pattern. A linear array consists of antenna elements arranged in a single dimension (straight line). A linear array of dipoles is shown in Figure 2. A standard spherical polar coordinate system is used, with the elements along the *x* axis. Broadside is $\theta = 0°$ and endfire is $\theta = \pm 90°$.

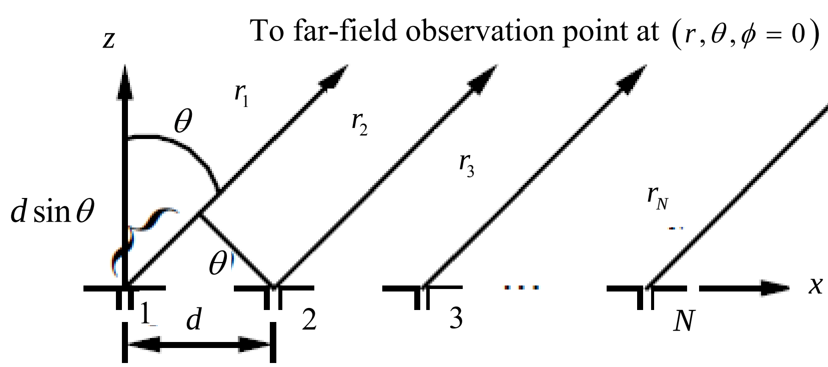


Figure 2. Linear array of dipoles (After [6]).

B. TWO-DIMENSIONAL ARRAYS WITH GROUND PLANES

A two-dimensional (2D) or planar array consists of antenna elements arranged on a two-dimensional plane. The theories and formulas for the development of the MATLAB program to synthesize the two-way antenna pattern for planar DSAs are discussed in this section. The material can be found in [7].

1. Array Factor

The array factor (AF) is a function of the array geometry and relative excitations of the array elements. Consider a planar array of point sources laid out on a rectangular lattice in the *x*-*y* plane as shown in Figure 3.

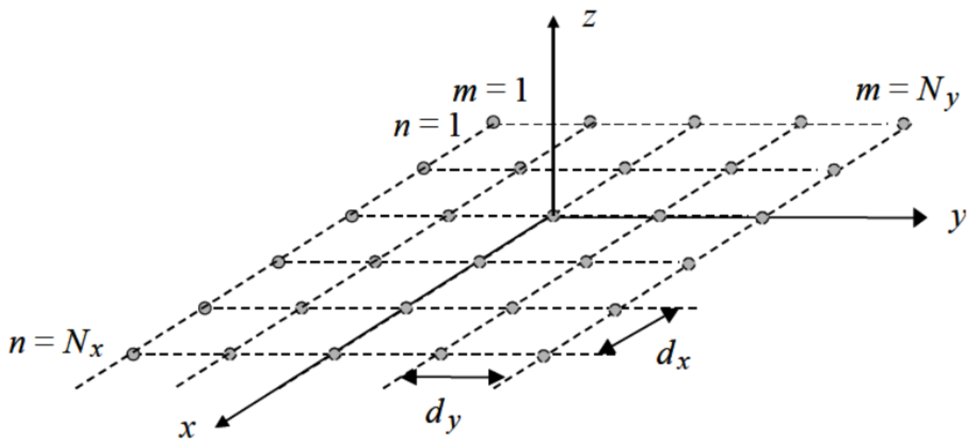


Figure 3. Planar array centered at the origin (From [7]).

The pattern angles θ and ϕ for the planar array oriented in the *x*-*y* plane are defined by the coordinate system shown in Figure 4.

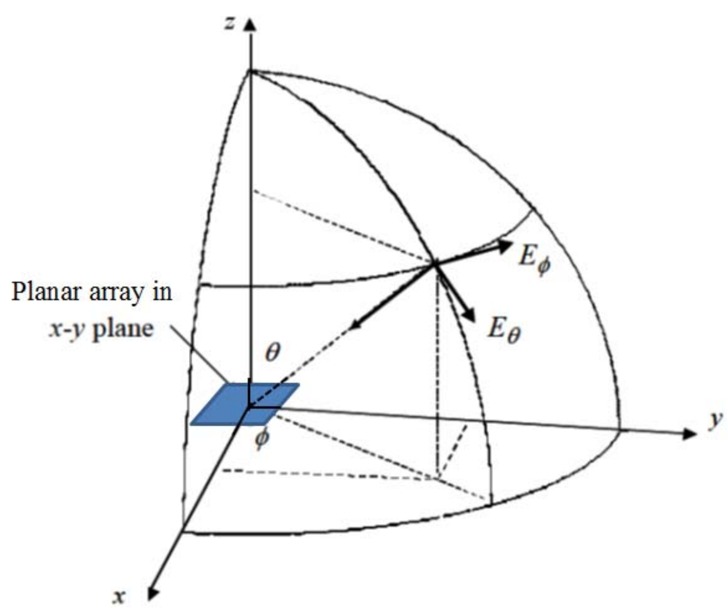


Figure 4. Array orientation and pattern angles θ and ϕ (After [8]).

If the array is centered at the origin, then the element locations can be written as

$$x_n = \frac{2n-\left(N_x+1\right)}{2} d_x, \quad n = 1, \ldots, N_x \tag{1}$$

and

$$y_m = \frac{2m-\left(N_y+1\right)}{2} d_y, \quad m = 1, \ldots, N_y \tag{2}$$

where d_x and d_y are the element spacings and N_x and N_y are the number of elements along the *x* and *y* axes, respectively. The array factor can be expressed as

$$AF = \sum_{n=1}^{N_x} \sum_{m=1}^{N_y} A_{mn} e^{j\beta x_n \sin\theta\cos\phi} e^{j\beta y_m \sin\theta\cos\phi} \tag{3}$$

where $\beta = 2\pi / \lambda$, and λ is the wavelength.

If the feeding arrangement results in a separable distribution (which is common), then $A_{mn} = A_m A_n$, and

$$AF = AF_x AF_y = \sum_{n=1}^{N_x} A_n e^{j\beta x_n \sin\theta\cos\phi} \sum_{m=1}^{N_y} A_m e^{j\beta y_m \sin\theta\cos\phi}. \tag{4}$$

Furthermore, if phases are introduced to scan the beam to the direction $\left(\theta_s, \phi_s\right)$, then

$$AF = \sum_{n=1}^{N_x} A_n e^{j\beta x_n \left(\sin\theta\cos\phi - \sin\theta_s\cos\phi_s\right)} \sum_{m=1}^{N_y} A_m e^{j\beta y_m \left(\sin\theta\cos\phi - \sin\theta_s\cos\phi_s\right)}. \tag{5}$$

Finally, if the array is uniformly excited $\left(A_{mn} = 1\right)$, the sum becomes a geometric series with the following closed form result [9]:

$$AF = AF_x AF_y = \frac{\sin\left(N_x \psi_x / 2\right)}{\sin\left(\psi_x / 2\right)} \frac{\sin\left(N_y \psi_y / 2\right)}{\sin\left(\psi_y / 2\right)} \tag{6}$$

where

$$\psi_x = \beta d_x \left(\sin\theta\cos\phi - \sin\theta_s\cos\phi_s\right) = \beta d_x \left(u - u_s\right)$$
$$\psi_y = \beta d_y \left(\sin\theta\sin\phi - \sin\theta_s\sin\phi_s\right) = \beta d_y \left(v - v_s\right) \tag{7}$$

and

$$u = \sin\theta\cos\phi$$
$$u_s = \sin\theta_s\cos\phi_s$$
$$v = \sin\theta\sin\phi \tag{8}$$
$$v_s = \sin\theta_s\sin\phi_s \, .$$

2. Ground Plane Factor

Ground planes are used typically to increase the directivity of the antenna array. Consider the array to be placed at a height *h* above a ground plane. If the ground plane is a perfect conductor of infinite extent, the method of images can be used to compute the field above the ground plane. The total array, including both the sources and their images, can be viewed as a planar array, where each element is a linear array along the *z*-axis as shown in Figure 5.

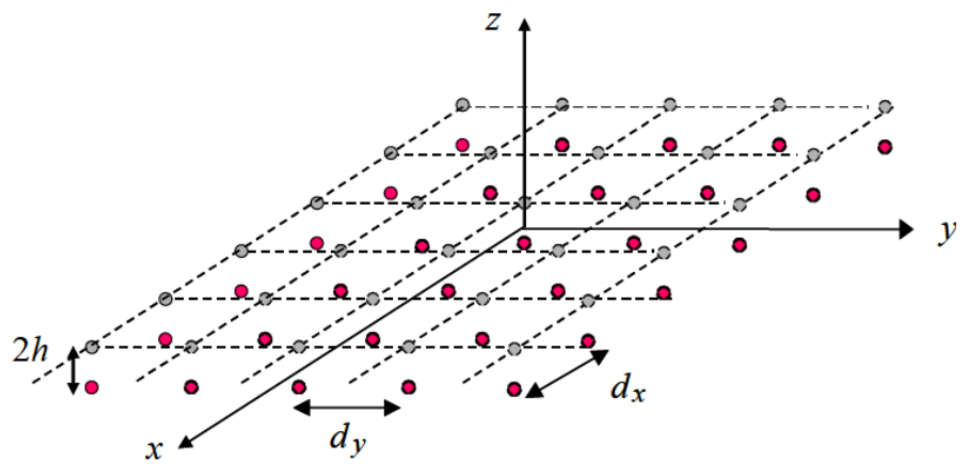


Figure 5. Two-element linear array along the *z*-axis, where each element is a planar array (From [7]).

With the images out of phase with the sources, the ground plane factor (GF) for the two element linear array along the *z*-axis can be written as

$$GF = e^{j\beta h\cos\theta} - e^{-j\beta h\cos\theta} = 2j\sin\left(\beta h\cos\theta\right). \tag{9}$$

3. Element Factor

The radiation behavior of a single antenna element is signified by its element pattern and can be characterized by its element factor. In general, the element factor has both θ and ϕ components. The element factors can be derived from the direction cosines (see Appendix A).

For a half-wave dipole along the *x*-axis with a maximum feed point current I_m, the element factors are

$$EF_\theta = \frac{j\eta_o I_m e^{-j\beta r}}{2\pi r}\left[\frac{\cos\left(\frac{\pi}{2}\sin\theta\cos\phi\right)}{1-\sin^2\theta\cos^2\phi}\right]\cos\theta\cos\phi \tag{10}$$

and

$$EF_\phi = -\frac{j\eta_o I_m e^{-j\beta r}}{2\pi r}\left[\frac{\cos\left(\frac{\pi}{2}\sin\theta\cos\phi\right)}{1-\sin^2\theta\cos^2\phi}\right]\sin\phi \tag{11}$$

where $\eta_o = 377\ \Omega$ is the intrinsic impedance of free space. For a half-wave dipole along the *y*-axis, the element factors are

$$EF_\theta = \frac{j\eta_o I_m e^{-j\beta r}}{2\pi r}\left[\frac{\cos\left(\frac{\pi}{2}\sin\theta\sin\phi\right)}{1-\sin^2\theta\sin^2\phi}\right]\cos\theta\sin\phi \tag{12}$$

and

$$EF_\phi = \frac{j\eta_o I_m e^{-j\beta r}}{2\pi r}\left[\frac{\cos\left(\frac{\pi}{2}\sin\theta\sin\phi\right)}{1-\sin^2\theta\sin^2\phi}\right]\cos\phi. \tag{13}$$

For a half-wave dipole along the *z*-axis, the element factors are

$$EF_\theta = \frac{j\eta_o I_m e^{-j\beta r}}{2\pi r}\left[\frac{\cos\left(\frac{\pi}{2}\cos\theta\right)}{\sin\theta}\right] \tag{14}$$

and

$$EF_\phi = 0\,. \tag{15}$$

To normalize the element factors, the leading factor $j\eta_o I_m e^{-j\beta r}/2\pi r$ is simply removed. If the dipoles are ideal (Hertzian with constant current and length of the dipoles $L \ll \lambda$) instead of half-wave dipoles, then the terms in square brackets [ ] become 1, and L is added to the leading factor. Therefore, after normalization of the element factors, only the trigonometric functions to the right of the bracketed terms remain.

4. Array Pattern

The total normalized pattern factor of the array is obtained using the principle of pattern multiplication:

$$\begin{aligned} F_{norm}(\theta,\phi) &= \left(AF_x \cdot AF_y\right)_{norm} \cdot GF_{norm} \cdot EF_{norm} \\ &= \left(AF_x \cdot AF_y \cdot GF \cdot EF\right)_{norm} \\ &= \frac{\sin\left(N_x\psi_x/2\right)}{N_x\sin\left(\psi_x/2\right)}\frac{\sin\left(N_y\psi_y/2\right)}{N_y\sin\left(\psi_y/2\right)}\sin\left(\beta h\cos\theta\right)EF_{norm} \end{aligned} \tag{16}$$

where EF_{norm} is the normalized element factor. Note that there are θ and ϕ components of F_{norm} (determined by the component of EF_{norm} used in Equation (16)).

C. GRATING LOBES

Considering a linear array along the *x*-axis, the array factor is a pattern that is symmetric about the axis of the array [9]. Therefore, the visible region is the region of the array factor that corresponds to $-90° \le \theta \le +90°$. The visible region is determined by the element spacing of the array in terms of wavelength. One period of the array factor appears in the visible region when the element spacing is one-half of the wavelength. When the element spacing exceeds one-half wavelength, more than one period of the array factor is visible and there may be more than one major lobe in the visible region. The different visible regions for different element spacing for an array factor with N=20 elements is shown in Figure 6.

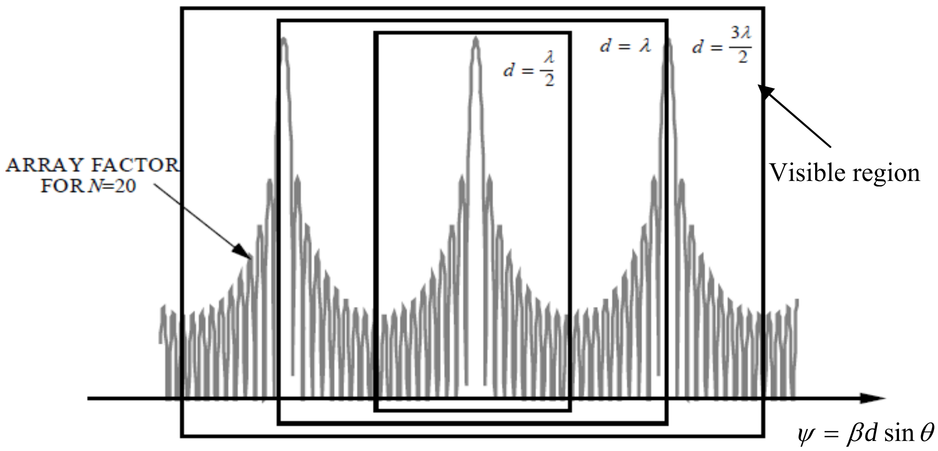


Figure 6. Visible regions for different element spacing d for N = 20 element array (After [6]).

These additional major lobes, which have peak intensities equal to that of the main lobe, are called grating lobes. The subarray spacing for a DSA is generally greater than one wavelength. Consider a linear DSA made up of M identical and equally-spaced subarrays separated by distance l_x with each subarray having N-element equally-spaced half-wave dipoles with element spacing d_x as shown in Figure 7.

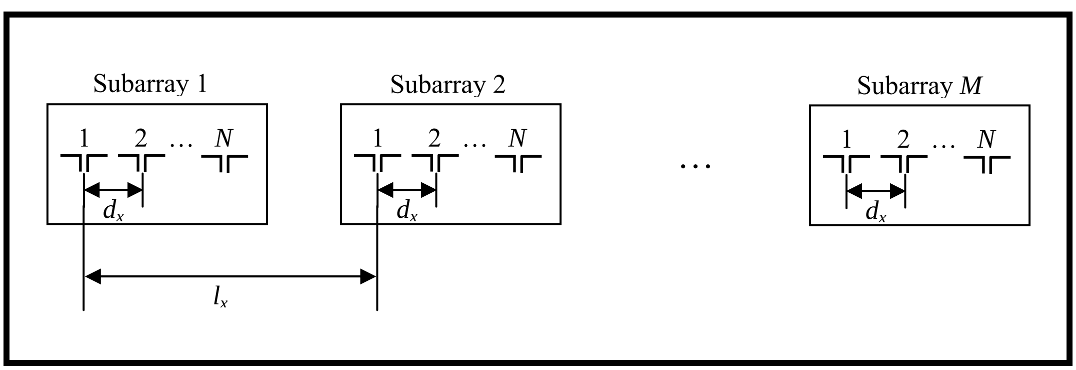


Figure 7. Linear arrangement of equally-spaced M subarrays with each subarray consisting of N-element equally-spaced half-wave dipoles.

The grating lobes produced by the DSA when $M = 5, N = 5, l_x = 7.5\lambda$ and $d_x = 1.5\lambda$ are shown in Figure 8. Broadside of the array is $\theta = 0°$. Grating lobes are located at about $\pm 42°$.

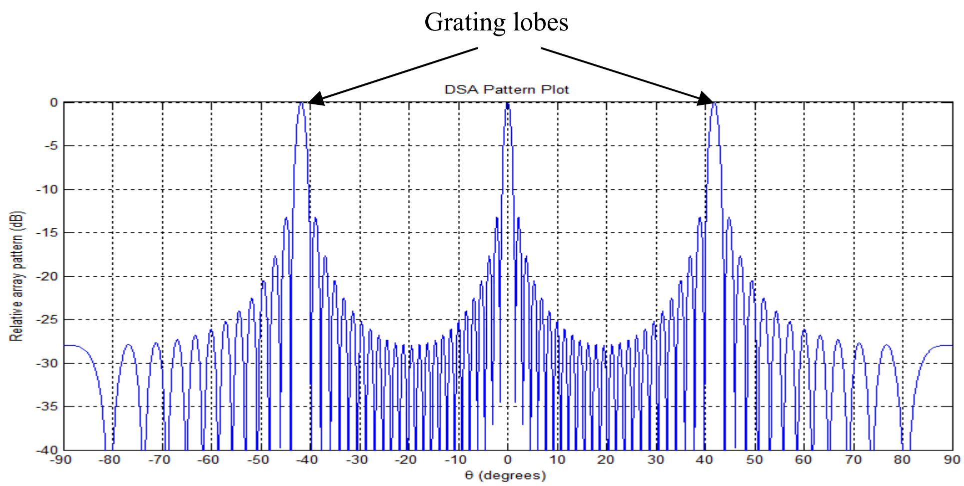


Figure 8. Grating lobes produced by linear DSA $(M = 5, N = 5, l_x = 7.5\lambda, d_x = 1.5\lambda)$.

If we consider a radar array antenna with the element and subarray spacing shown in Figure 8, it will not be able to distinguish a target at an angle of $+42°$ from one at its broadside ($\theta = 0°$). Therefore, the radar may be unable to associate the target with the correct angle, creating angular ambiguity. Another problem caused by grating lobes is the unwanted clutter echoes that are picked up by the grating lobes. The result is a reduction in the signal-to-clutter ratio which can limit target detection [10].

As grating lobes are generally the result of the periodicity associated with widely-spaced identical contiguous subarrays, one approach to suppress grating lobes is to implement subarrays of unequal sizes, with random locations of the subarray centers [11]. Another approach is to implement overlapping-subarray architecture to push the grating lobes away from the main beam and shape the subarray patterns in order to suppress the grating lobes in the subarray's low side lobe region [12]. Another possible method to suppress grating lobes is to use separate transmit and receive array architectures. This method is based on the principle of pattern multiplication for the two-way pattern design.

The grating lobes in the transmit pattern can be suppressed or eliminated by placing nulls in the receive pattern positions coincident with the transmit grating lobes or vice versa.

D. SUMMARY

The fundamental theory of array antennas was presented in this chapter. The array pattern was obtained using the principle of pattern multiplication of the element factor, array factor, and ground plane factor. A two-dimensional array above a ground plane was considered. Grating lobes are produced when either the element spacing or subarray spacing is more than one-half wavelength. In most radar applications, it is undesirable to have grating lobes as they can cause angular ambiguities and excess background clutter. The subarray spacing for a DSA is generally of multiple wavelengths, and thus, grating lobes will exist. Therefore, it is of interest to suppress these grating lobes in DSA design. In the next chapter, the radar range equation for DSA, directivity and gain are introduced. The approach to suppress undesired grating lobes using the principle of pattern multiplication is also discussed.

THIS PAGE INTENTIONALLY LEFT BLANK

III. DISTRIBUTED PHASED ARRAYS

The principle of digital beamforming, the radar range equation for DSA, directivity, and gain are presented in this chapter. Undesired grating lobes produced by widely-spaced subarrays can be suppressed by using the principle of pattern multiplication. This approach is discussed in this chapter.

A. BEAMFORMING IN PHASED ARRAYS

Electronic scanning or beamforming in phased arrays is achieved by varying the amplitude and phase between antenna elements. A conventional phased array usually employs an analog beamforming network consisting of microwave transmission lines and power dividers. These are physically large and heavy if there are a large number of antenna elements, especially for ground or shipboard phased arrays. With the advent of digital processing, modern phased arrays may use digital beamforming networks. A phased array that employs a digital beamformer is called digital phased array. A linear array with digital beamforming on receive is shown in Figure 9.

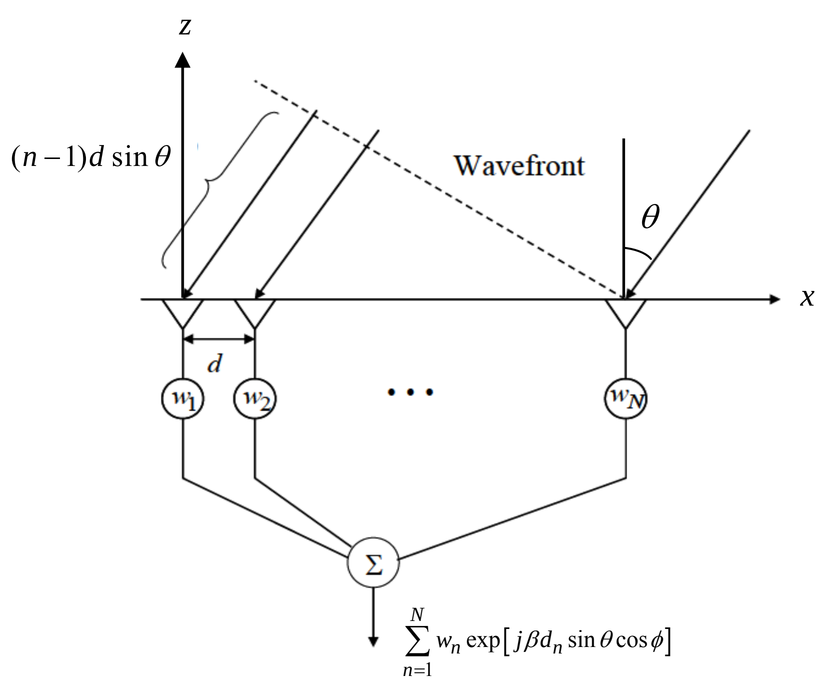


Figure 9. Digital array beamforming on receive (After [13]).

The array factor for the linear array in Figure 9 in the *x*-*z* plane $(\phi = 0°)$ is given by

$$AF(\theta,\phi) = \sum_{n=1}^{N} w_n \exp\left[j\beta d_n \sin\theta \cos\phi \right] = \sum_{n=1}^{N} A_n e^{j\alpha_n} \exp\left[j\beta d_n \sin\theta \cos\phi \right] \quad (17)$$

where N is the number of elements, $w_n = A_n e^{j\alpha_n}$ is a complex weight added by the processor, and d_n is given by Equation (1).

If quadrature demodulators are used at each element, they provide spatial samples of the exponential factor in Equation (17):

$$I_n = \cos\left[j\beta d_n \sin\theta \cos\phi \right] \quad (18)$$

$$Q_n = \sin\left[j\beta d_n \sin\theta \cos\phi \right]. \quad (19)$$

Therefore, to obtain a response equivalent to a beam scanned in the direction (θ_s, ϕ_s), the weights must be given by

$$w_n = \exp\left[-j\beta d_n \sin\theta_s \cos\phi_s \right]. \quad (20)$$

B. RADAR RANGE EQUATION FOR DISTRIBUTED SUBARRAYS

The radar range equation (RRE) for distributed radar systems is derived in [4] and also summarized in [14]. Similarly, we can extend the RRE derivation to radars employing DSAs if we consider the general geometry of a DSA system as shown in Figure 10.

The following assumptions are made to simplify the derivation:

- The effects due to multipath are neglected.
- The mutual coupling between subarrays and subarray elements is neglected.
- The signals are of one polarization.
- The case of monostatic scattering from the target applies.

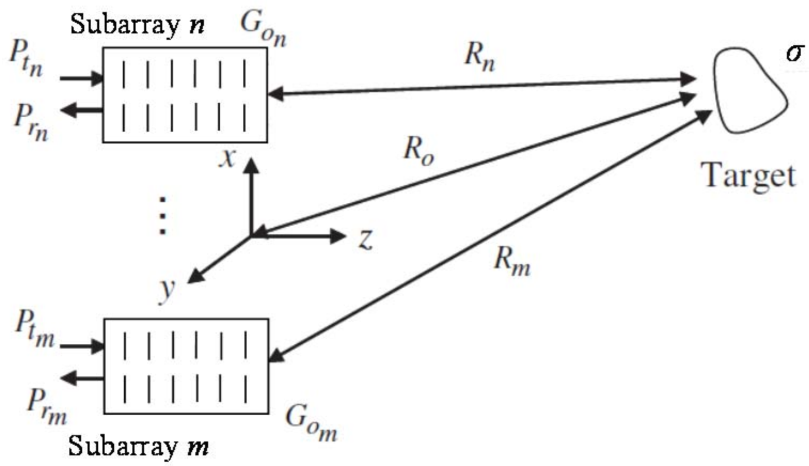


Figure 10. General geometry of distributed subarray system (After [14]).

If the target is in the far-field with respect to the DSA, the time-averaged (denoted by an over-bar) scattered power density from the target at receive subarray *m* when subarray *n* is transmitting is given by [1]

$$\bar{W}_{mn}^{s} = \left(\frac{\bar{P}_{t_n} G_{o_n}}{4\pi R_n^2} \right) \left(\frac{\sigma}{4\pi R_m^2} \right) \tag{21}$$

where $\bar{P}_{t_n}$ is the time-averaged power transmitted by subarray *n*, and G_{o_n} is the antenna gain of subarray *n*. Note that this includes the element and ground plane factor for subarray *n*. The parameter σ is the RCS of the target (assuming monostatic scattering).

The amplitude of the peak phasor electric field at subarray *m* is obtained from the scattered power density

$$\bar{W}_{mn}^{s} = \frac{\left| E_{mn}^{s} \right|^2}{2\eta_o} . \tag{22}$$

The phase of the electric field at receive subarray *m* is determined by the path length $(R_n + R_m)$. Therefore, the scattered electric field received by subarray *m* due to transmitting subarray *n* is

$$E_{mn}^{s} = \sqrt{\frac{2\eta_o \bar{P}_{t_n} G_{o_n} \sigma}{(4\pi)^2 R_n^2 R_m^2}} \exp\left\{ j\left[-\beta\left(R_n + R_m\right) + \psi_{t_n} \right] \right\} \tag{23}$$

where ψ_{t_n} is the phase added to transmitting subarray *n* to scan and focus the collective array beam.

Hence, for N_s transmitting subarrays, the total scattered electric field at the receiving subarray *m* is given by

$$E_m^s = \sum_{n=1}^{N_s} E_{mn}^s = \sum_{n=1}^{N_s} \sqrt{\frac{2\eta_o \bar{P}_{t_n} G_{o_n} \sigma}{(4\pi)^2 R_n^2 R_m^2}} \exp\left\{ j\left[-\beta\left(R_n + R_m\right) + \psi_{t_n} \right] \right\}. \tag{24}$$

The total received power can be obtained by multiplying the incident power density by the effective area of the receive subarray *m*, A_{e_m}. Since the relationship between gain and effective area [9] is

$$A_{e_m} = \frac{G_{o_m} \lambda^2}{4\pi} \tag{25}$$

we can write the total received power as

$$\bar{P}_{r_m} = \frac{G_{o_m} \lambda^2}{(4\pi)^3} \left| \sum_{n=1}^{N_s} \sqrt{\frac{\bar{P}_{t_n} G_{o_n} \sigma}{R_n^2 R_m^2}} \exp\left\{ j\left[-\beta\left(R_n + R_m\right) + \psi_{t_n} + \psi_{r_m} \right] \right\} \right|^2 \tag{26}$$

where ψ_{r_m} is the phase added to the receive channel to focus and scan the beam.

Since the case of monostatic scattering is considered and we can assume that equal phase focusing is used to achieve coherence on the target, we can make the following substitutions:

- For the case of monostatic scattering: $R_n = R_m \equiv R_o$.
- If the beam is focused on target: $\psi_{t_n} + \psi_{r_m} = \beta\left(R_n + R_m\right) \equiv 2kR_o$.

For a single receive subarray *m*, the total received power can be simplified to

$$\overline{P}_{r_m} = \frac{G_{o_m} \lambda^2 \sigma}{(4\pi)^3 R_o^4} \left| \sum_{n=1}^{N_s} \sqrt{\overline{P}_{t_n} G_{o_n}} \right|^2 . \tag{27}$$

Therefore, if N_s transmit subarrays and M_s receive subarrays are used, the total power received by the DSA is given by

$$\overline{P}_r = \sum_{m=1}^{M_s} \overline{P}_{r_m} = \frac{\lambda^2 \sigma}{(4\pi)^3 R_o^4} \sum_{m=1}^{M_s} G_{o_m} \left| \sum_{n=1}^{N_s} \sqrt{\overline{P}_{t_n} G_{o_n}} \right|^2 . \tag{28}$$

C. DIRECTIVITY AND GAIN

The directivity of an array antenna is defined as the ratio of the radiation intensity in a certain direction to the average radiation intensity [9]. The maximum directivity is given as

$$D(\theta,\phi) = \frac{4\pi}{\Omega_A} \left| F_{norm}(\theta,\phi) \right|^2 \tag{29}$$

where Ω_A is the beam solid angle [9] and is defined as

$$\Omega_A = \int_0^{2\pi} \int_0^{\pi/2} \left| F_{norm}(\theta,\phi) \right|^2 \sin\theta d\theta d\phi \tag{30}$$

and F_{norm} is the normalized pattern factor of the total array. Note that the normalized subarray pattern for each subarray is of the form of Equation (16). Substituting (30) into (29), we get

$$D(\theta,\phi) = \frac{4\pi}{\int_0^{2\pi} \int_0^{\pi/2} \left| F_{norm}(\theta,\phi) \right|^2 \sin\theta d\theta d\phi} \left| F_{norm}(\theta,\phi) \right|^2 . \tag{31}$$

From Equation (31), we observe that the directivity is determined by the total radiation pattern of the antenna. The power gain measures how efficient the antenna array is in transforming input power to radiated power and is related to directivity by

$$G(\theta,\phi) = \varepsilon D(\theta,\phi) \tag{32}$$

where ε is the efficiency of the antenna array. Since the antenna effective aperture A_e [9] is related to beam solid angle by

$$A_e = \varepsilon \frac{\lambda^2}{\Omega_A}, \tag{33}$$

we can write the power gain of the antenna array as

$$G(\theta,\phi) = \frac{4\pi A_e}{\lambda^2} \left| F_{norm}(\theta,\phi) \right|^2 . \tag{34}$$

D. PATTERN MULTIPLICATION

When a DSA is made up of identical subarrays periodically arranged with equal spacing between subarrays, the total pattern of the DSA is simply the multiplication of the subarray pattern and the configuration pattern determined by the arrangement of the subarrays [5]. For a planar DSA in the *x*-*y* plane composed of M_x by M_y uniformly excited subarrays with spacing l_x and l_y, the normalized pattern factor of the total array is

$$F_{norm}(\theta,\phi) = F_{norm_S}(\theta,\phi) \times AF_c(\theta,\phi) \tag{35}$$

where $F_{norm_S}(\theta,\phi)$ is the normalized pattern factor of the subarray from (16) and

$$AF_c = AF_{x_c} AF_{y_c} = \frac{\sin(M_x \xi_x / 2)}{\sin(\xi_x / 2)} \frac{\sin(M_y \xi_y / 2)}{\sin(\xi_y / 2)} \tag{36}$$

where

$$\begin{aligned} \xi_x &= \beta l_x \left(\sin\theta \cos\phi - \sin\theta_s \cos\phi_s \right) \\ \xi_y &= \beta l_y \left(\sin\theta \sin\phi - \sin\theta_s \sin\phi_s \right). \end{aligned} \tag{37}$$

Since the geometric arrangement of the subarrays is uniform, the grating lobes of the total array pattern can be predicted. The angular directions of the grating lobes [10] in the DSA configuration pattern are located at

$$\sin\theta\cos\phi - \sin\theta_s\cos\phi_s = \pm\frac{\lambda}{l_x}p \quad (38)$$

and

$$\sin\theta\sin\phi - \sin\theta_s\sin\phi_s = \pm\frac{\lambda}{l_y}p \quad (39)$$

where $p, q = 0, 1, 2, \ldots, \infty$. Note that grating lobe locations do not depend on the number of elements in the subarray.

Similarly, the null locations of the uniform subarray pattern can be found at

$$\sin\theta\cos\phi - \sin\theta_s\cos\phi_s = \pm\frac{\lambda}{N_x d_x}p \quad (40)$$

and

$$\sin\theta\sin\phi - \sin\theta_s\sin\phi_s = \pm\frac{\lambda}{N_y d_y}q \quad (41)$$

except for *p*, *q* integer multiples of N_x, N_y, respectively.

From (40) and (41), it is observed that the locations of nulls depend on the number of elements in the subarray. Observing (38) and (39), we see that the condition in which the grating lobes of the DSA configuration pattern coincide with the nulls of the subarray pattern is when $l_x / d_x = N_x$ and $l_y / d_y = N_y$. However, this is the condition for contiguous subarray [12]. In order to suppress some ratio of the grating lobes in the DSA configuration pattern and still have widely-spaced subarrays congruent to DSA design, the subarray-to-element spacing ratio and corresponding number of elements per subarray need to be correctly chosen. A simple program written using MATLAB can be used to plot out the grating lobe and null locations in the direction cosine space for a selected number of elements in a subarray, the element spacing, and subarray spacing to help visualize the suppression of grating lobes of the configuration pattern using coincident nulls in the subarray pattern. The MATLAB codes for the program are given in Appendix B. Using an example of a transmit array with the configuration $N_x = N_y = 5,\ d_x = d_y = 0.5\lambda,\ l_x = l_y = 5\lambda$, the grating lobe and null locations in direction cosine (u, v) space are shown in Figure 11. In this example, the scanning angles are set to $\theta_s = 0°$ and $\phi_s = 0°$. From Figure 11, we observe that the even-numbered grating lobes caused by the subarray spacing are suppressed by coincident nulls from the subarray pattern. Other grating lobes which do not have nulls that are coincident in the u, v space are not suppressed.

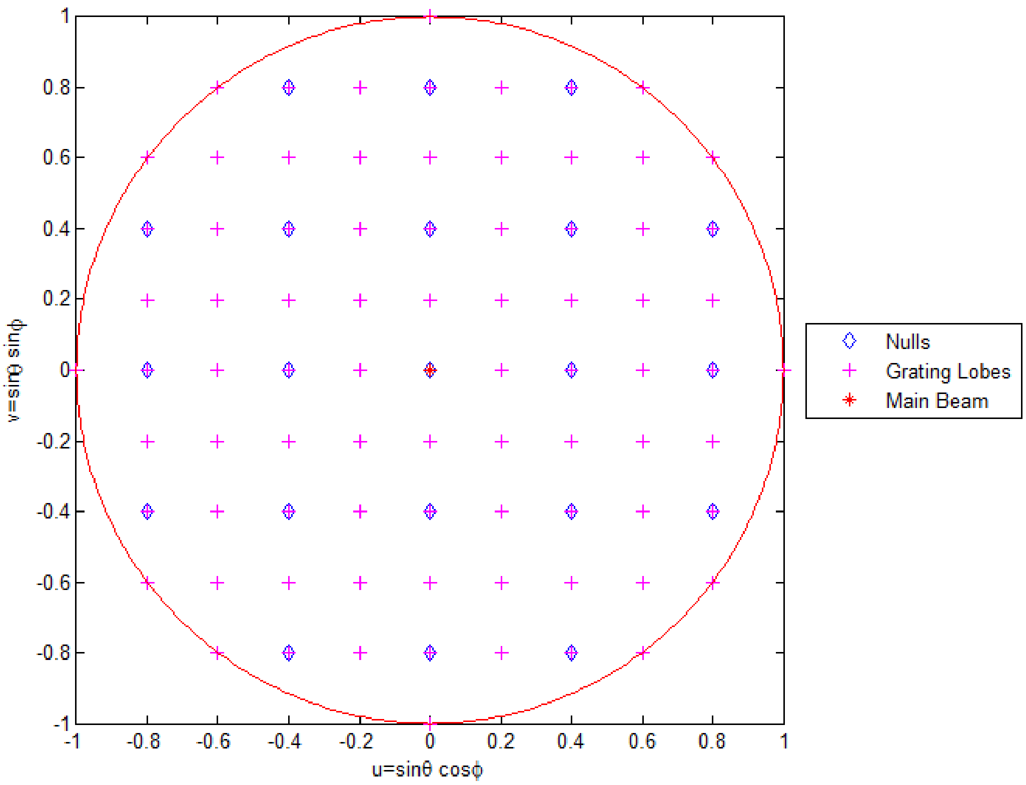


Figure 11. Grating lobes and null locations for transmit array with $N_x = N_y = 5,\ d_x = d_y = 0.5\lambda,\ l_x = l_y = 5\lambda,\ \theta_s = \phi_s = 0°$.

By applying the principle of pattern multiplication, the normalized two-way pattern is defined as

$$F_{norm_{2way}}(\theta,\phi) = F_{norm_{Tx}}(\theta,\phi) \times F_{norm_{Rx}}(\theta,\phi) \tag{42}$$

where $F_{norm_{Tx}}(\theta,\phi)$ is the normalized total pattern of transmit array and $F_{norm_{Rx}}(\theta,\phi)$ is the normalized total pattern of receive array. Using the two-way pattern multiplication approach, we see that the remaining grating lobes can be suppressed with nulls from the receive array. If the receive array is configured as a contiguous array such that $N_x = N_y = 10,\ d_x = d_y = 0.5\lambda,\ l_x = l_y = 5\lambda$, the grating lobe and null locations will be coincident as shown in Figure 12.

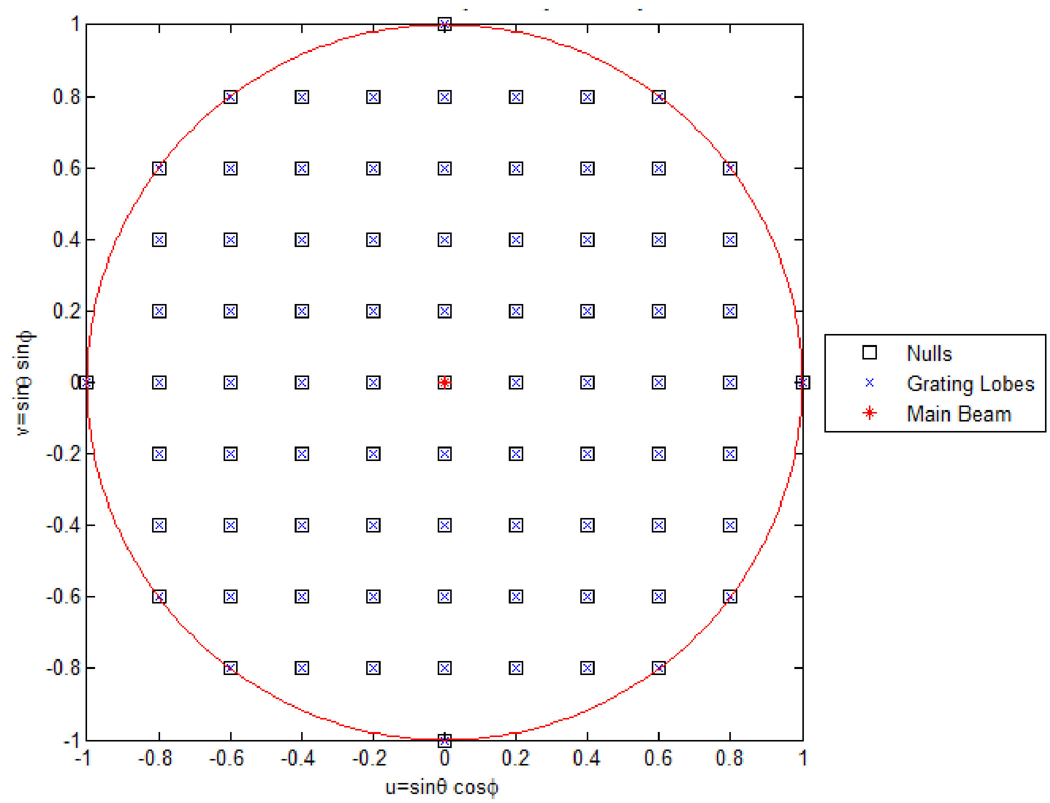


Figure 12. Grating lobes and null locations for receive array with $N_x = N_y = 10,\ d_x = d_y = 0.5\lambda,\ l_x = l_y = 5\lambda\ ,\ \theta_s = \phi_s = 0°$.

Overlapping the plots for the transmit and receive arrays, it is observed that the locations of correctly placed nulls of the receive array are coincident with the remaining grating lobes from the transmit array as shown in Figure 13. Therefore, the remaining grating lobes can be suppressed.

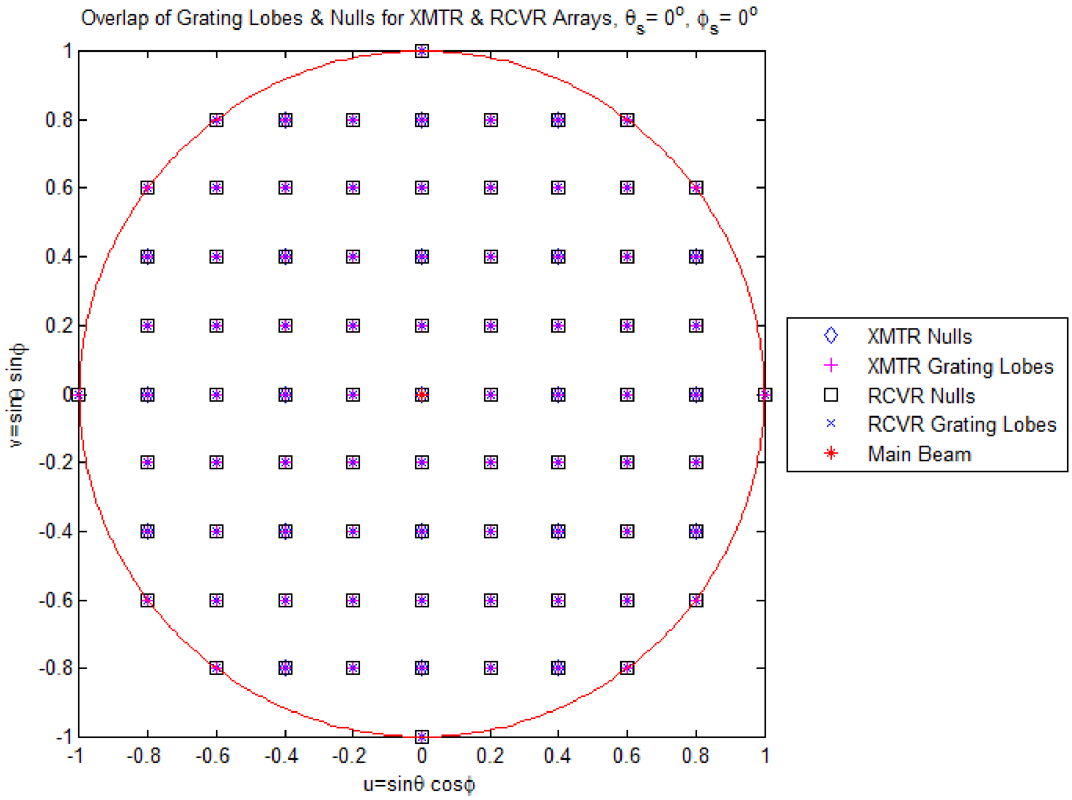


Figure 13. Overlap of the grating lobes and null locations for transmit and receive arrays.

E. SUMMARY

In this chapter, the principles of digital beamforming, directivity, and gain for phased arrays were presented, and the RRE for radar systems employing DSAs was also examined. It is understood that undesired grating lobes produced by widely-spaced subarrays can be suppressed using the principle of pattern multiplication by intentional placement of nulls coincident to grating lobe locations. To visualize the placement of grating lobes and nulls for transmit and receive array patterns, their locations are plotted in direction cosine space using a simple program developed in MATLAB. This forms the basis of two-way pattern synthesis in deciding the optimum element spacing, subarray spacing, and number of elements in each subarray, for the transmit and receive arrays.

In the next chapter, a simulation tool developed to perform the two-way pattern calculations for DSAs is described. The results of using the tool to simulate the two-way patterns from different DSA configurations is presented and analyzed as well.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. TWO-WAY PATTERN DESIGN FOR DSA

The simulation tool developed to perform the two-way pattern calculation of transmit and receive DSAs is described in this chapter. The analysis and simulation results obtained using the simulation tool for different DSA configurations are also presented.

A. SIMULATION TOOL FOR TWO-WAY PATTERN

The ***Two-way Antenna Pattern Calculation for Distributed Subarrays*** program is a simulation tool developed and implemented in MATLAB to perform the calculation and display of transmit, receive, and two-way patterns for user configured DSAs. The graphical user interface (GUI) of the program menu is shown in Figure 14.

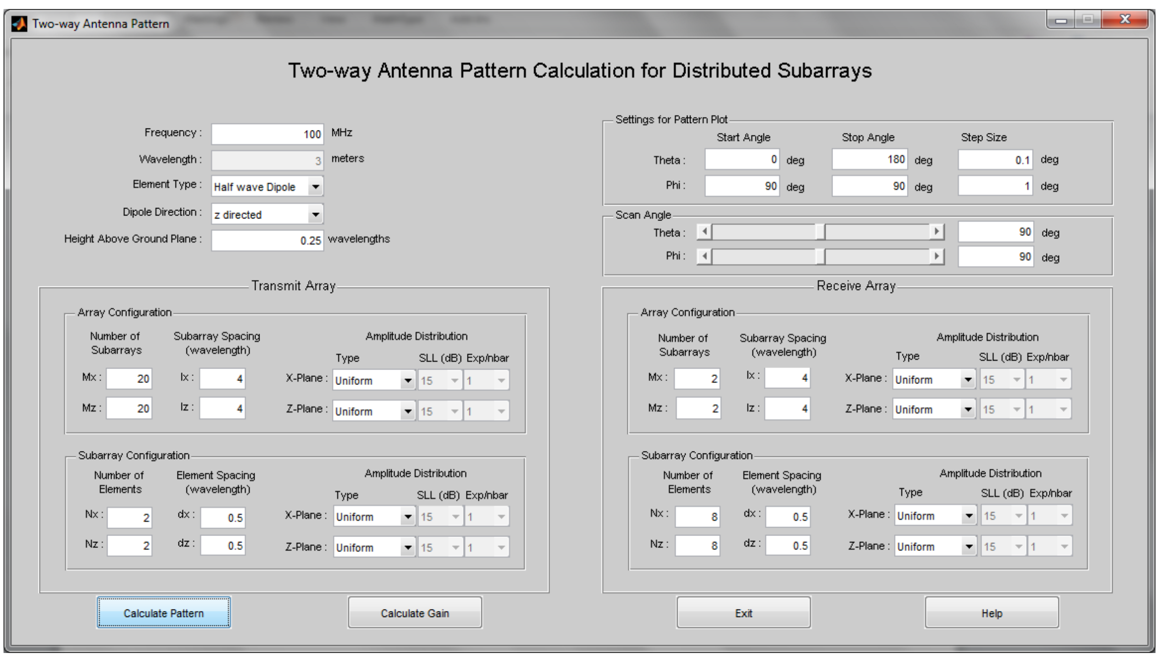


Figure 14. GUI of program main menu.

Note that the array is in the *x*-*z* plane with the *y*-axis normal to the aperture as defined by the coordinate system shown in Figure 15. The ground plane is in the *x*-*z* plane and the array is considered to be placed at a height above the ground plane (in the positive *y* direction).

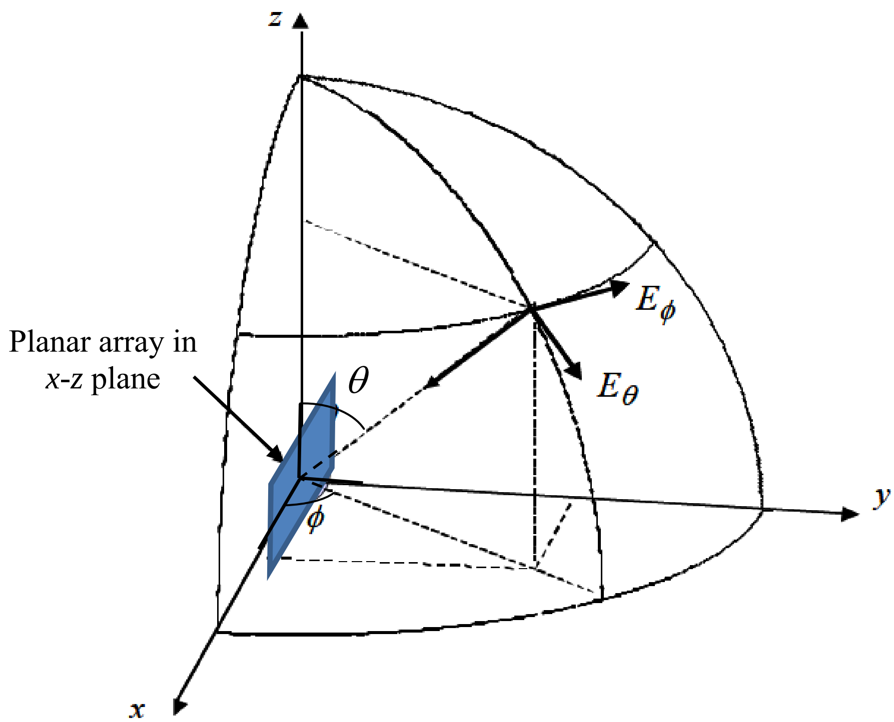


Figure 15. Coordinate system with planar array in the *x*-*z* plane orientation and pattern angles θ and ϕ (After [8]).

To run the program GUI, the user will need to run MATLAB and set the path or directory to where the program and its subroutine codes are stored. The program directory should contain the following MATLAB files:

- *two_way_pattern.m*
- *two_way_pattern.fig*
- *helpfig.m*
- *helpfig.fig*
- *dsaplot.m*
- *caf_hdip2*.m
- *caf_sdip2.m*
- *caf_iso2.m*
- *getamplitudes.m*
- *bayliss.m*
- *cosine.m*

- *tayl.m*
- *compute_gain.m*
- *gausq20.m*

The MATLAB codes and description of the program files are given in Appendix C. After setting the path or directory, type in *two_way_pattern* at the MATLAB command line and the program GUI will be executed.

1. DSA Parameter Inputs

The program GUI provides a convenient way to key in the DSA parameters for different transmit/receive configurations. The required inputs are classified into basic parameters, pattern plot range settings, scan angle settings, transmit DSA configuration settings, and receive DSA configuration settings.

a. Basic Parameters

The basic parameters that the user needs to enter in the program GUI are frequency (in MHz), element type, dipole direction, and the height of the elements above the ground plane. The wavelength in meters is automatically calculated by the program when the array's frequency information is entered. The user is allowed to select the element types: half-wave dipole, short dipole, or isotropic element from a pull-down list. When the half-wave dipole or short dipole is selected, the program allows the user to choose either *x*-directed (parallel) or *z*-directed (collinear) element orientations. The user is able to set the height of the array elements above a ground plane in terms of wavelengths.

b. Pattern Plot Range and Step Size

The program computes the DSA pattern plot over the maximum range $0° \le \theta \le 180°$ and $0° \le \phi \le 180°$. However, the user can configure that range of angles for the pattern calculation under the GUI panel labeled *Settings for Pattern Plot*. The user is allowed to change the start angle, stop angle, and step size for θ and ϕ in degrees. The range and step size settings are applied to the transmit DSA, receive DSA, and two-way pattern calculation. If the start and stop values for θ are identical, then a pattern cut for that value of θ is generated and displayed. Similarly, if the start and stop values for ϕ are identical, then a pattern cut for that value of ϕ is generated and displayed. If a range of both θ and ϕ is given, then a three-dimensional mesh plot is generated. The step size setting allows the user to enter an appropriate step interval for θ and ϕ angles. A small step size gives fine resolution data plots but requires more computation time, while a large step size requires less time to compute but produces coarse data plots. The default value for the step size is 3°.

c. Beam Scanning Angles

The array scan angle refers to the steering of the main beam by phase control. The user is able to set the scan angles for both θ and ϕ independently for the range $0° \le \theta \le 180°$ and $0° \le \phi \le 180°$. The default scan angles are set to $90°$.

d. Transmit/Receive DSA Configuration Settings

Two similar array GUI panels in the main menu allow the transmit and receive DSAs to be configured independently. Within each panel, the user is allowed to configure the number of subarrays (M_x, M_z) and spacing (l_x, l_z) in both x and z planes as well as select the desired amplitude distribution. The default amplitude distribution is *uniform,* but a pull-down list box allows the user to choose from *taylor*, *cosine*, *bayliss,* and *triangular* amplitude distributions. In the case of *taylor*, *cosine,* and *bayliss* amplitude distributions, the user is able to select the arbitrary sidelobe level (SLL). The number of elements (N_x, N_z) and element spacing (d_x, d_z) in both x and z planes that form a subarray can be configured. Similarly, the amplitude distribution for the elements in the subarray can also be set.

2. Two-Way DSA Pattern Calculation

After the parameters for the transmit and receive DSAs are set, the user can select the *Calculate Pattern* button on the program GUI to initiate the computation of the array patterns of the transmit and receive DSAs as well as the two-way pattern. The process flow diagram to compute and plot the array patterns is shown in Figure 16. The flow diagram of the sub-process to calculate the DSA one-way array pattern for half-wave dipoles is shown in Figure 17.

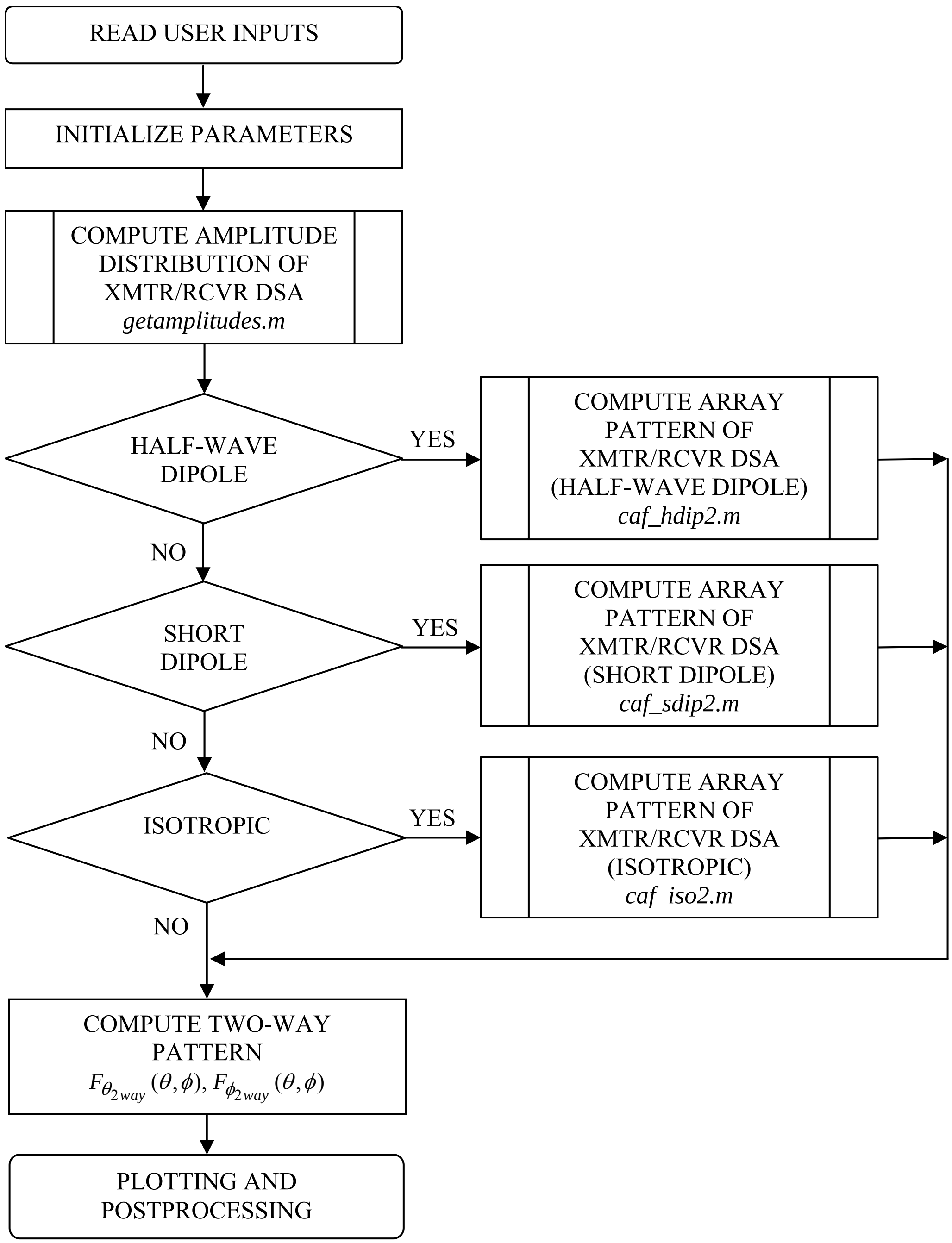


Figure 16. Flow diagram for calculating two-way pattern of DSAs.

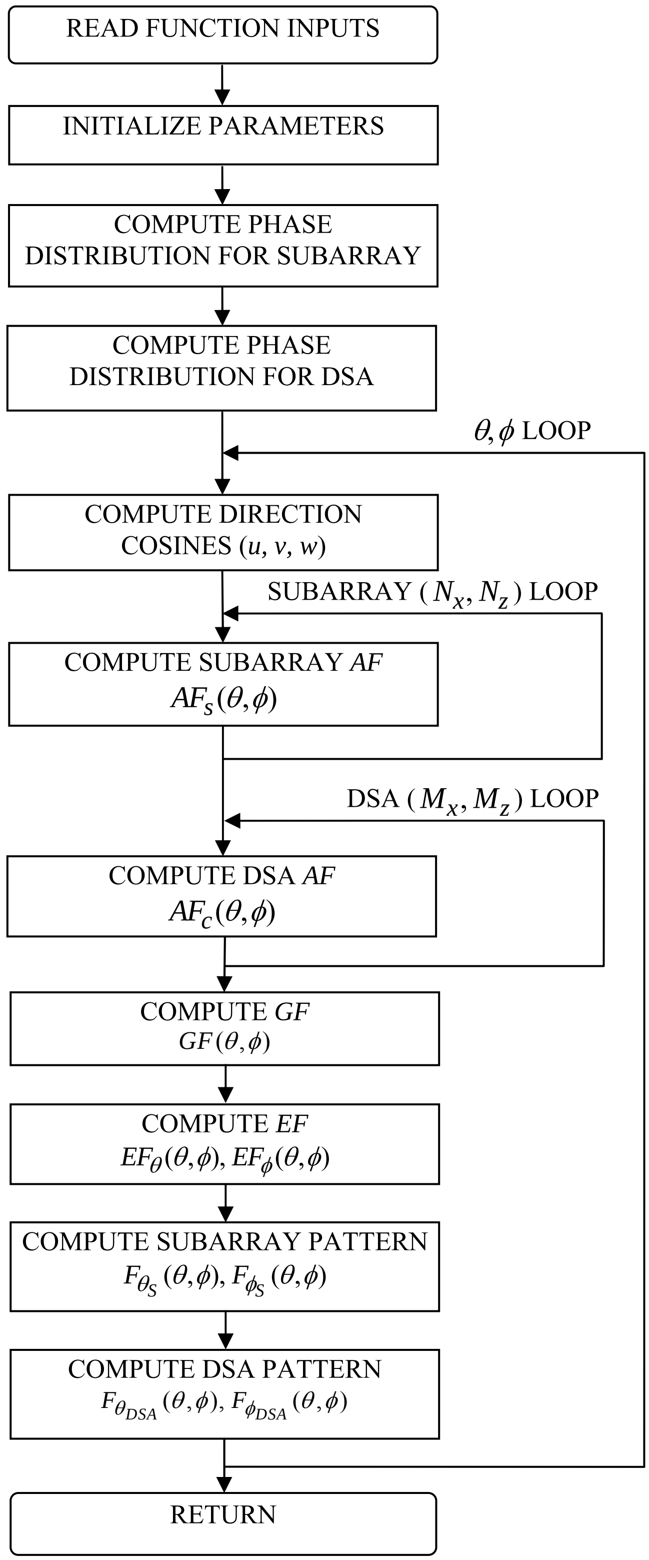


Figure 17. Flow diagram of function *caf_hdip2.m* to calculate DSA one-way pattern for half-wave dipole elements.

3. Output Plots for DSA Patterns

When the start and stop values for θ are identical, the program generates and displays the rectangular plots for the normalized transmit DSA, receive DSA, and two-way pattern cut for that value of θ in both θ and ϕ components. Similarly, if the start and stop values for ϕ are identical, the program generates and displays the rectangular plots for the normalized transmit DSA, receive DSA, and two-way pattern cut for that value of ϕ in both θ and ϕ components. For the configuration settings from Figure 14, the rectangular plots for the normalized transmit DSA, receive DSA, and two-way pattern cut for $\phi = 90°$ in θ component are illustrated in Figures 18, 19 and 20 respectively. When a range of both θ and ϕ are given, the program produces normalized three-dimensional mesh plots for the transmit, receive, and two-way patterns of the DSAs in both θ and ϕ components. Using similar configuration settings from Figure 14, but setting the pattern plot range for both θ and ϕ from $0°$ to $180°$ and the step size $1°$, we get the output mesh plot for the two-way pattern for the θ component in Figure 21. The mesh plot is given in direction cosine (u, w) space.

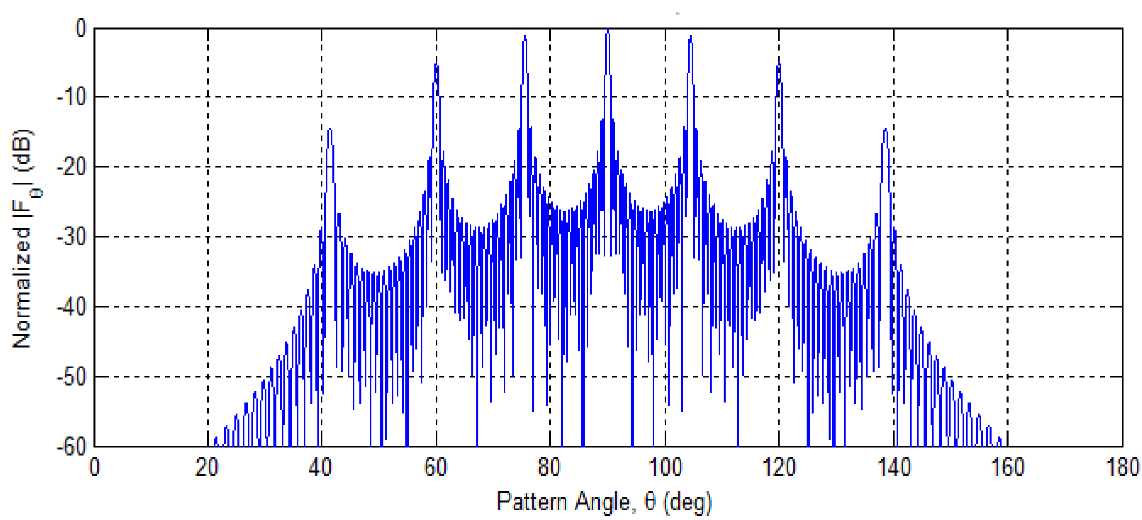


Figure 18. Normalized transmit DSA pattern for θ component, $\phi = 90°$ pattern cut.

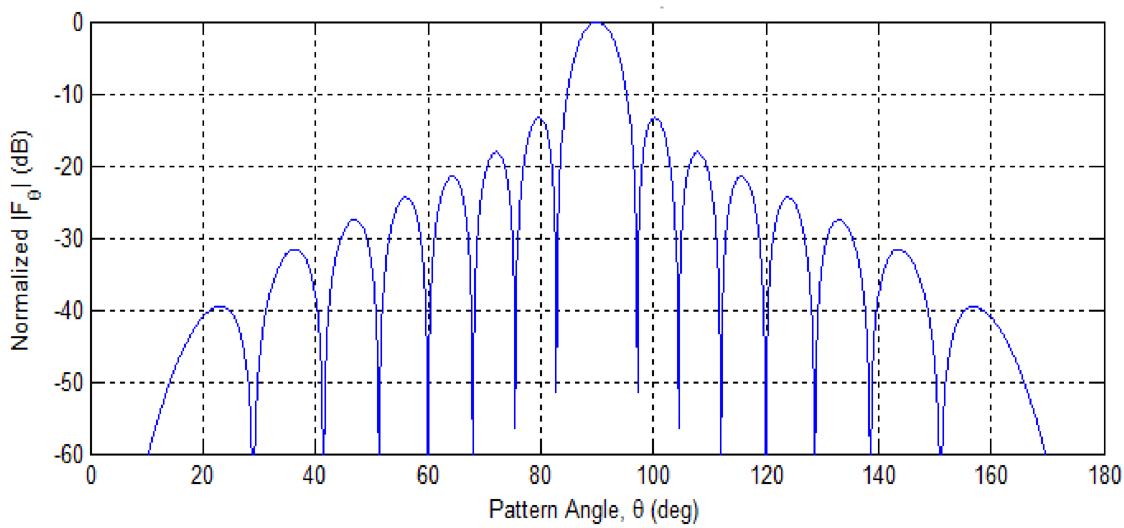


Figure 19. Normalized receive DSA pattern for θ component, $\phi = 90°$ pattern cut.

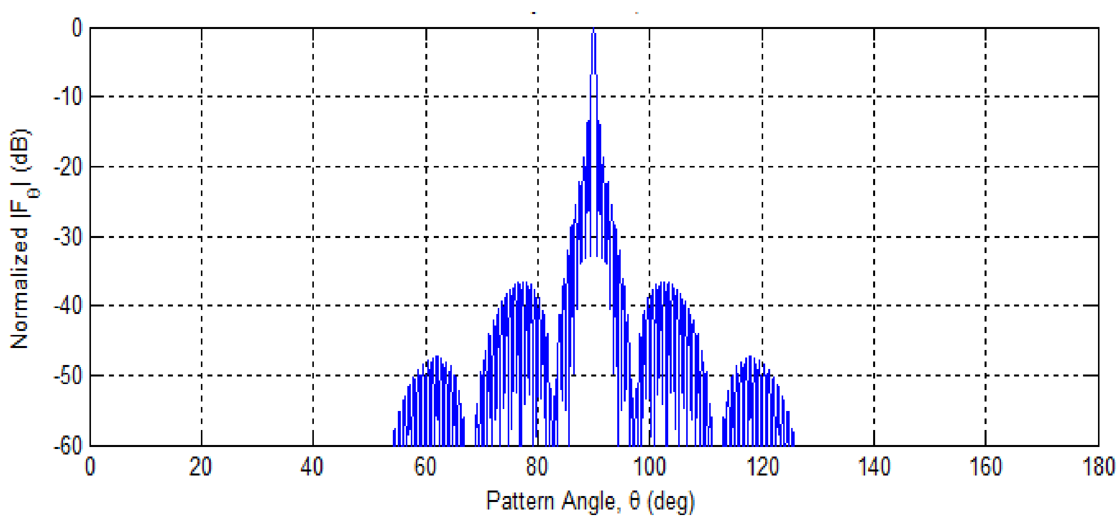


Figure 20. Normalized two-way pattern for θ component, $\phi = 90°$ pattern cut.

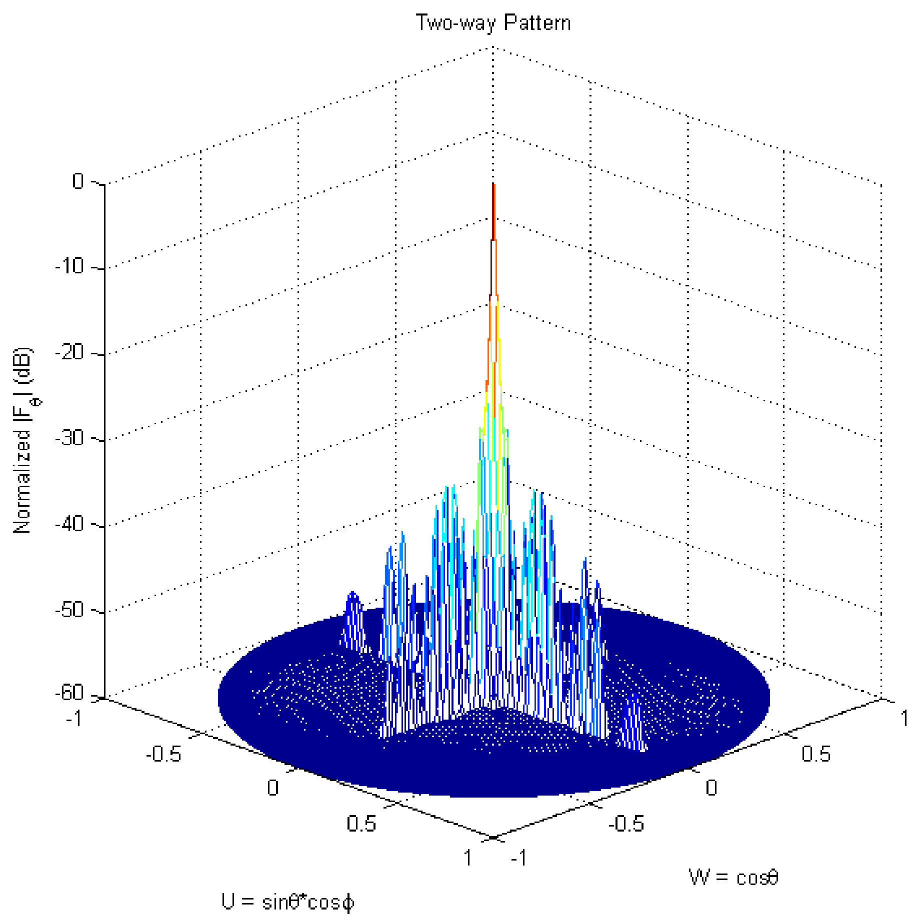


Figure 21. Example of mesh plot of normalized two-way pattern for θ component.

4. Gain Calculation

The user can select the *Calculate Gain* button on the program GUI to initiate the gain calculation of the transmit and receive DSAs as well as the two-way pattern. The computation of gain is performed based on the parameters configured on the program GUI. Gain calculation is performed numerically using N_θ and N_ϕ intervals of Gaussian quadrature integration, with 20 points per interval, over the range $0° \le \theta \le 180°$ and $0° \le \phi \le 180°$. The integration constants are loaded from MATLAB file *gausq20.m*. The user is required to set the number of integration intervals for the range of θ and ϕ in a user input dialog box after the *Calculate Gain* button is selected. The user input dialog box is shown in Figure 22. The user should enter positive integer values only. The total number of integration points is the product of the number of integration intervals multiplied by 20. For example, if the integration interval for θ is set to four, the total number of integration points is 80 over the range $0° \le \theta \le 180°$. If there are also four intervals in ϕ, the total number of integration points is $80 \times 80 = 6400$. A large integration interval produces more accurate gain results but increases the computation time.

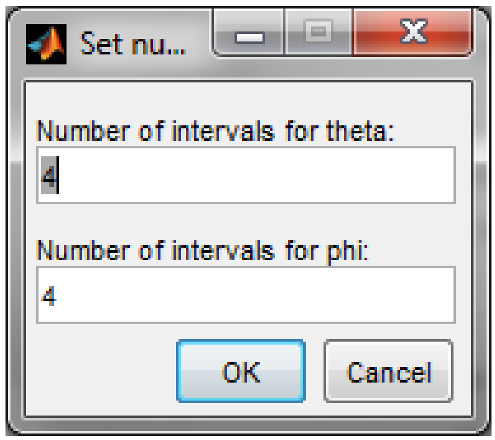


Figure 22. User input dialog box to set integration intervals for θ and ϕ for gain calculation.

The process flow diagram for two-way gain calculation is shown in Figure 23. The program calls a sub-process from MATLAB function file *compute_gain.m* to calculate the one-way gain of the transmit DSA and receive DSA. The flow diagram of the sub-process for one-way gain calculation is shown in Figure 24.

After the computation is completed, power gain results for the transmit DSA, receive DSA, and two-way pattern are displayed in both numeric and decibel format. The power gain results computed for the configuration settings in Figure 14 are shown in Figure 25. An integration interval value six is used for both θ and ϕ.

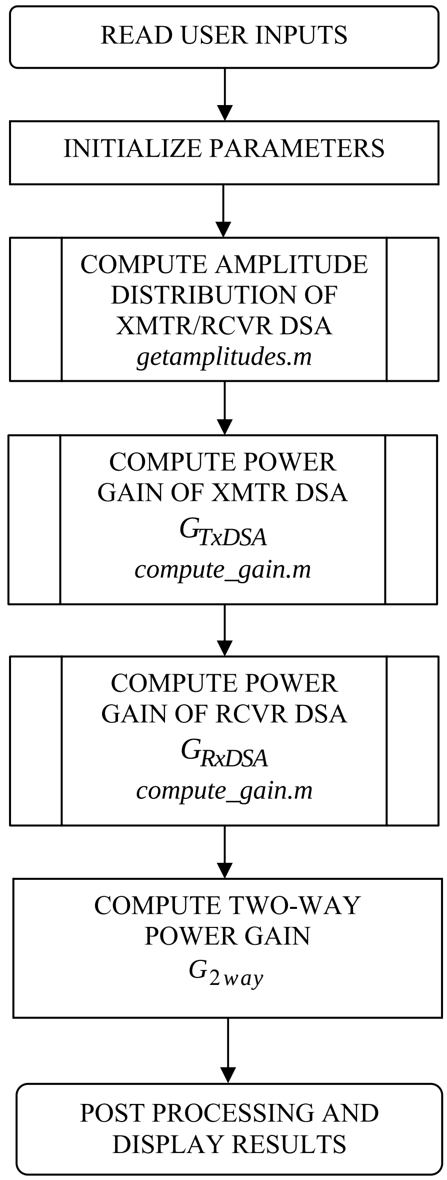
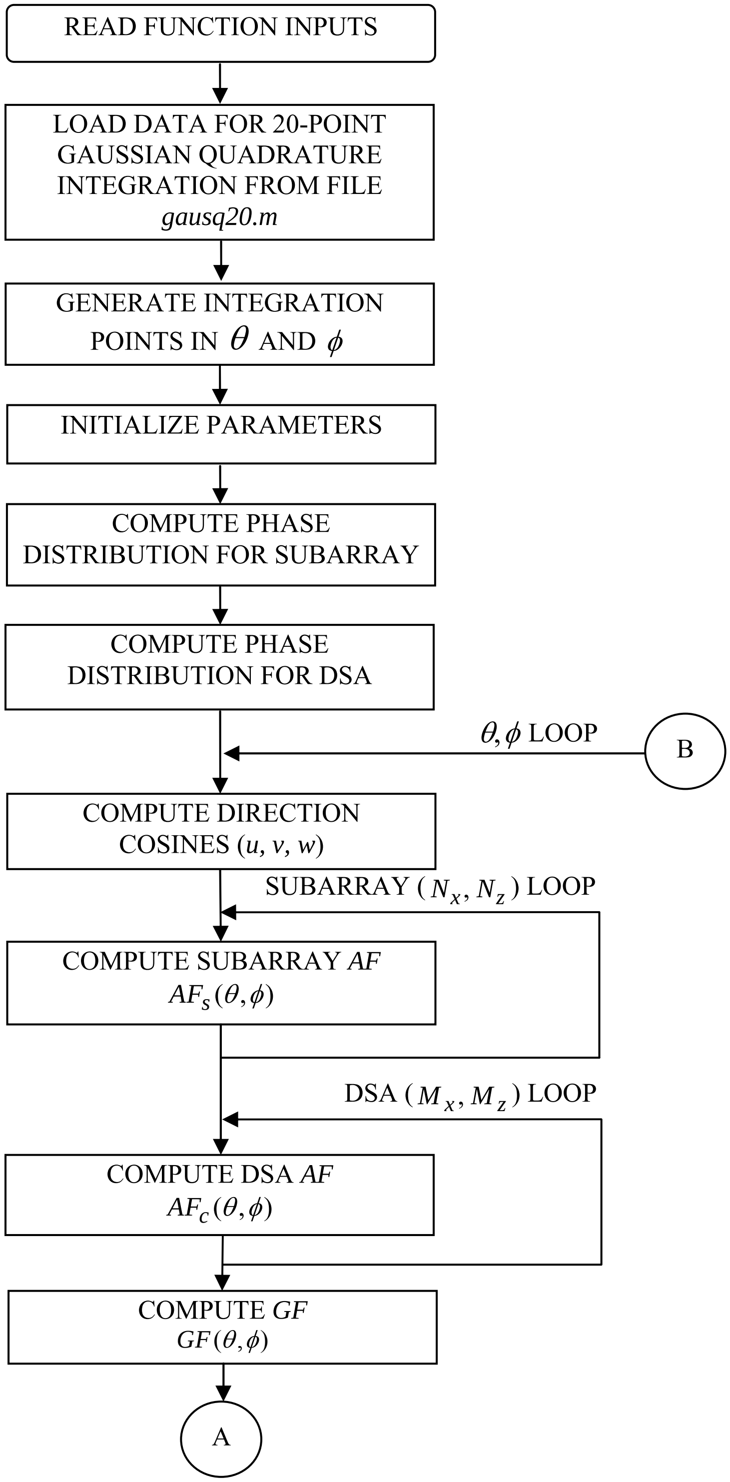


Figure 23. Flow diagram for calculating two-way power gain of DSAs.


READ FUNCTION INPUTS
LOAD DATA FOR 20-POINT GAUSSIAN QUADRATURE INTEGRATION FROM FILE *gausq20.m*
GENERATE INTEGRATION POINTS IN θ AND ϕ
INITIALIZE PARAMETERS
COMPUTE PHASE DISTRIBUTION FOR SUBARRAY
COMPUTE PHASE DISTRIBUTION FOR DSA
θ, ϕ LOOP
B
COMPUTE DIRECTION COSINES (*u, v, w*)
SUBARRAY (N_x, N_z) LOOP
COMPUTE SUBARRAY *AF* $AF_s(\theta,\phi)$
DSA (M_x, M_z) LOOP
COMPUTE DSA *AF* $AF_c(\theta,\phi)$
COMPUTE *GF* $GF(\theta,\phi)$
A

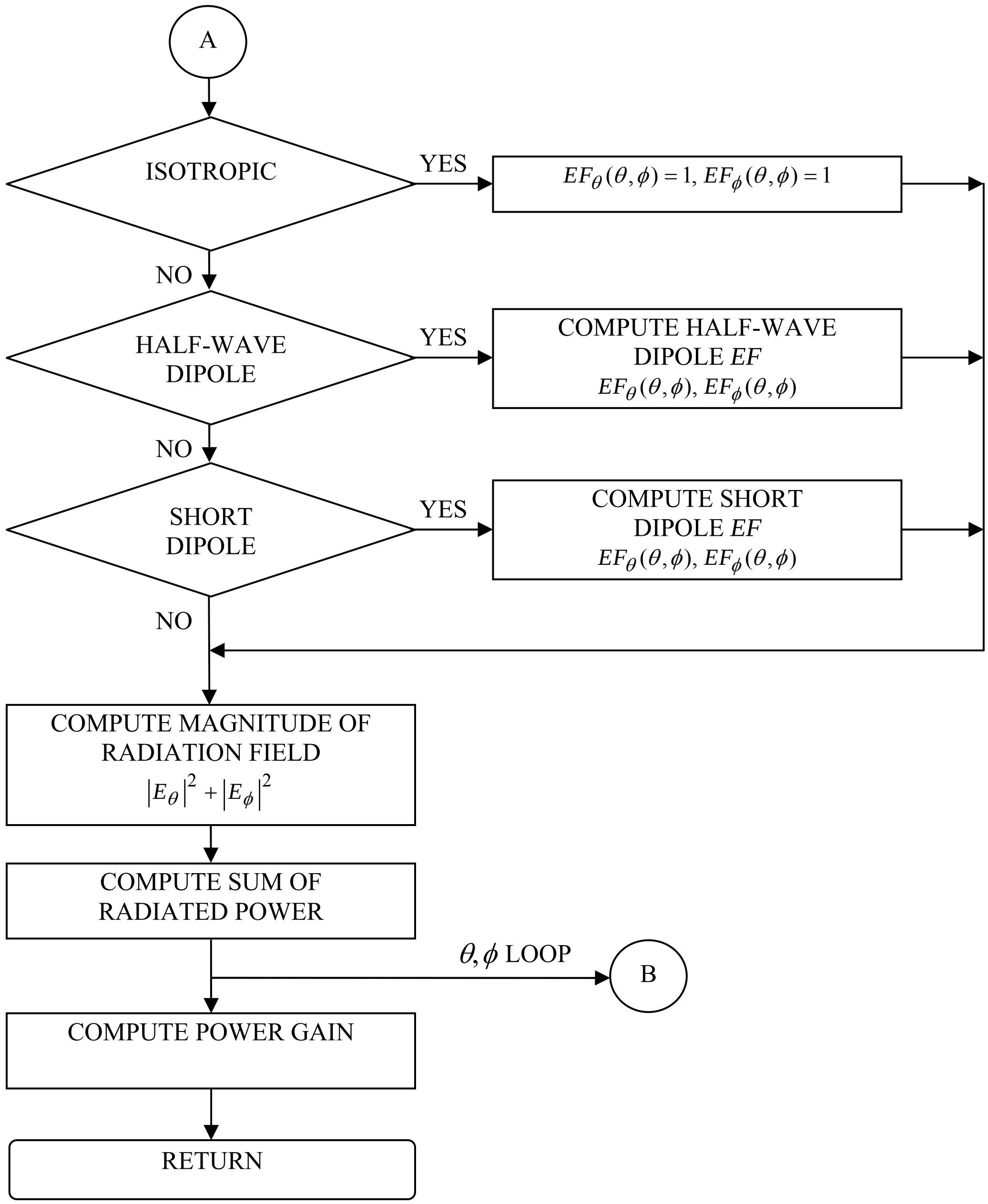


Figure 24. Flow diagram of function *compute_gain.m* to calculate power gain for a DSA.

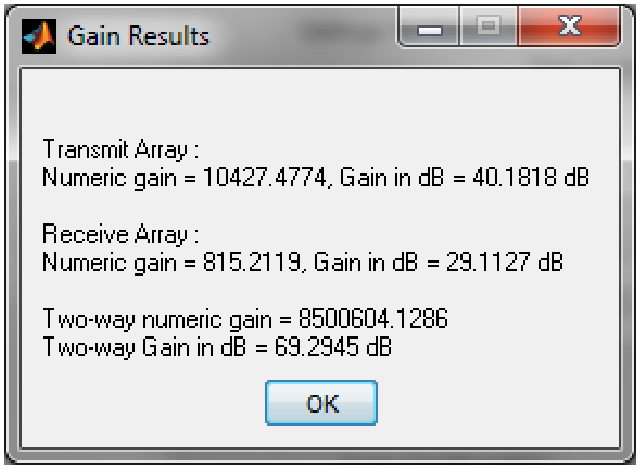


Figure 25. Example of power gain results dialog window.

5. Pattern and Configuration Data Files

After the computation of the array patterns, the results in complex form and the configuration settings are auto-saved to MATLAB binary data format files *dsapattern.mat* and *dsaconfig.dat*, respectively, in the program directory. These files allow the user to extract the pattern data and configuration settings easily in MATLAB. The files are overwritten when a new pattern calculation process is initiated.

B. DSA SIMULATIONS AND RESULTS

To illustrate the suppression of grating lobes by placement of coincident nulls using the principle of pattern multiplication, the example of transmit and receive array configuration settings from Figure 26 is used. An isotropic element is selected as the antenna element type, and the transmit and receive arrays are placed at a height of 0.25 λ above a ground plane. Uniform amplitude distribution is used for both transmit and receive arrays with beam scan angles $\theta_S = \phi_S = 90°$ (i.e., no beam scan). The transmit antenna consists of DSAs, and the separate receive antenna is configured to be contiguous designed with an array pattern that has nulls in the direction of the transmit array's grating lobes.

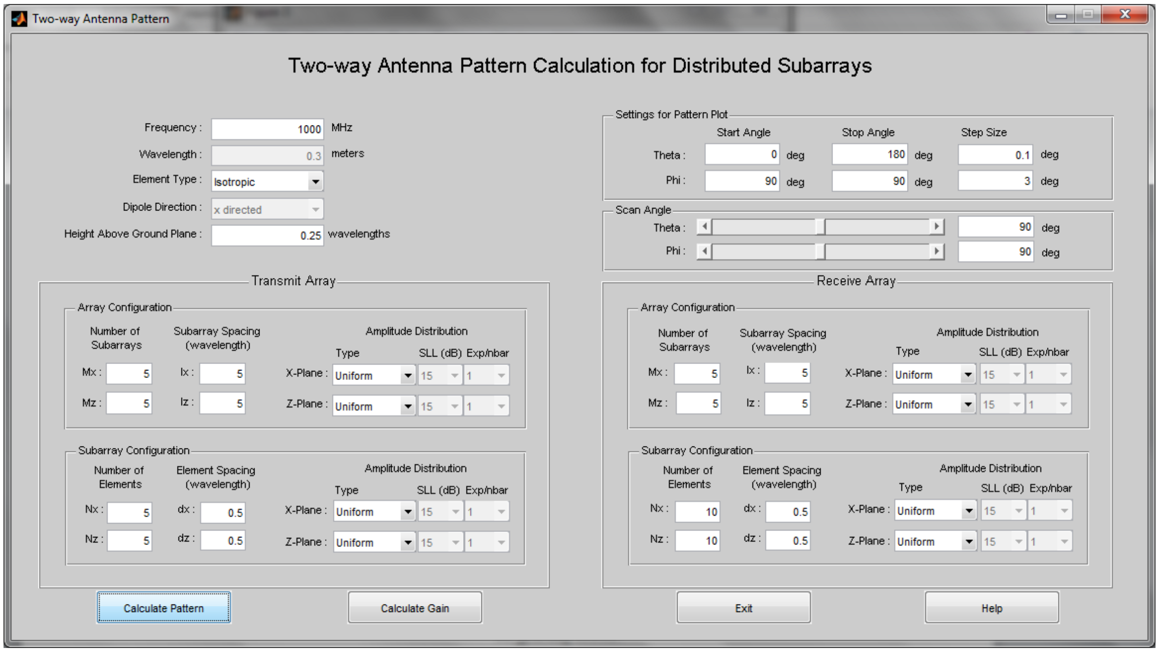


Figure 26. Configuration settings for transmit and receive DSAs to show suppression of grating lobes.

Using the configuration $N_x = N_z = 5, d_x = d_z = 0.5\lambda, M_x = M_z = 5, l_x = l_z = 5\lambda$ for the transmit array, we have $l_x / d_x = 2N_x$ and $l_z / d_z = 2N_z$. Using the simulation tool, we obtain the normalized transmit array pattern for the θ component with pattern angle range $0° \le \theta \le 180°$, $\phi = 90°$ cut shown in Figure 27. From Figure 27, we observe that the even-numbered grating lobes due to subarray spacing at θ pattern angles 36.8°, 66.4°, 113.6° and 143.2° are suppressed by coincident nulls from the subarray pattern. However, the grating lobes at θ pattern angles 11°, 53.2°, 78.7°, 101.3°, 126.8° and 169° are not suppressed. The peak SLL due to grating lobes is observed to be −3.7 dB. From Figure 27, the beamwidth between first nulls (BWFN) of the transmit array is approximately 4.6°. The half-power beamwidth (HPBW) of the transmit array is approximately 2°. The three-dimensional mesh plot and contour plot for the pattern angles $0° \le \theta \le 180°$ and $0° \le \phi \le 180°$ of the transmit array pattern for the θ component are shown in Figures 28 and 29, respectively. The step size for both θ and ϕ of the pattern plots are set to 0.3°. From the mesh and contour plots, we are able to observe the grating lobe locations in direction cosine (u, w) space.

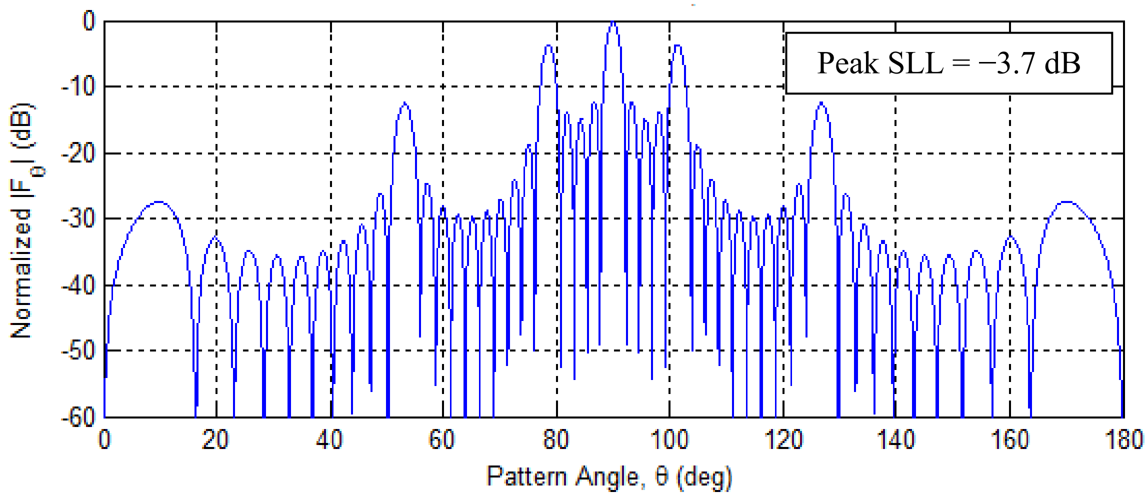


Figure 27. Normalized transmit array pattern for the θ component, $\phi = 90°$ cut with configuration settings $N_x = N_z = 5, d_x = d_z = 0.5\lambda, M_x = M_z = 5, l_x = l_z = 5\lambda$.

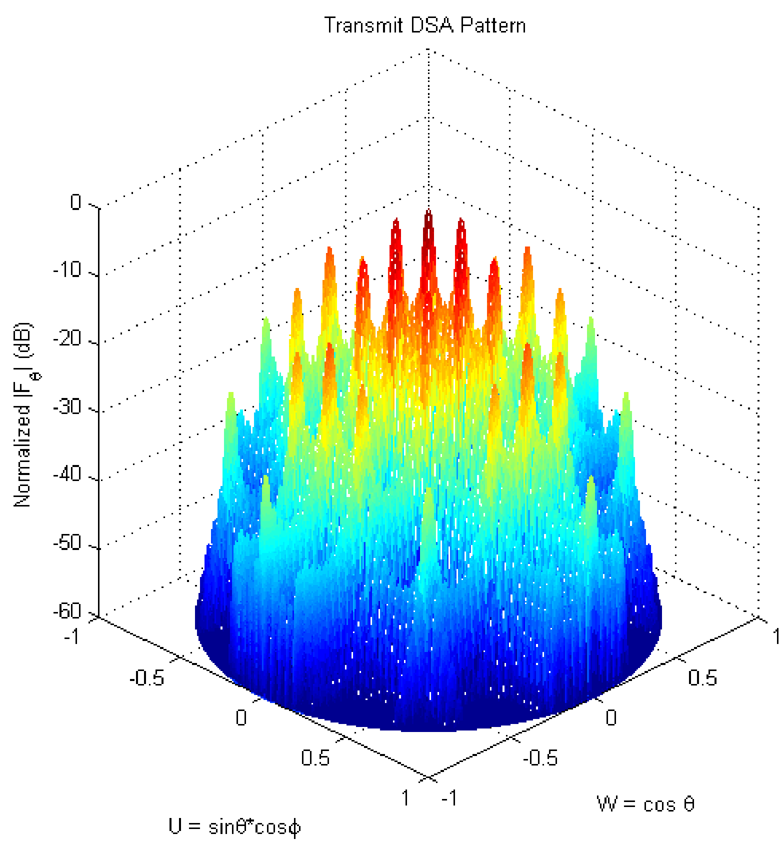


Figure 28. Normalized mesh plot of transmit array pattern for the θ component with configuration settings $N_x = N_z = 5, d_x = d_z = 0.5\lambda, M_x = M_z = 5, l_x = l_z = 5\lambda$.

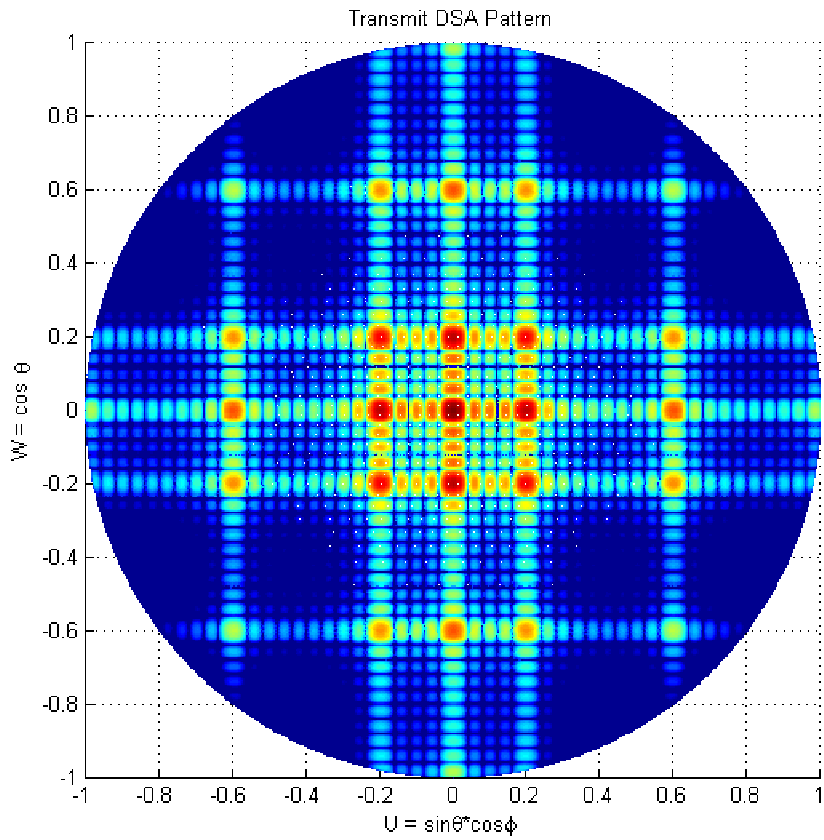


Figure 29. Normalized contour plot of transmit array pattern for the θ component with configuration settings $N_x = N_z = 5, d_x = d_z = 0.5\lambda, M_x = M_z = 5, l_x = l_z = 5\lambda$.

To place nulls at the remaining grating lobe locations of the transmit array, we can apply a contiguous subarray configuration for the receive array with the settings given in Figure 26. The normalized receive array pattern for the θ component with pattern angle range $0° \le \theta \le 180°$, $\phi = 90°$ cut is shown in Figure 30. Since the receive array is contiguous with element spacing 0.5λ, no grating lobes exist in the visible region. The peak SLL is observed to be −13.3 dB. The BWFN of the receive array is approximately 4.6°, and HPBW of the receive array is approximately 2°. The three-dimensional mesh plot and contour plot for the pattern angles $0° \le \theta \le 180°$ and $0° \le \phi \le 180°$ of the receive array pattern for the θ component are shown in Figures 31 and 32, respectively. The step size for both θ and ϕ of the pattern plots are set to 0.3°.

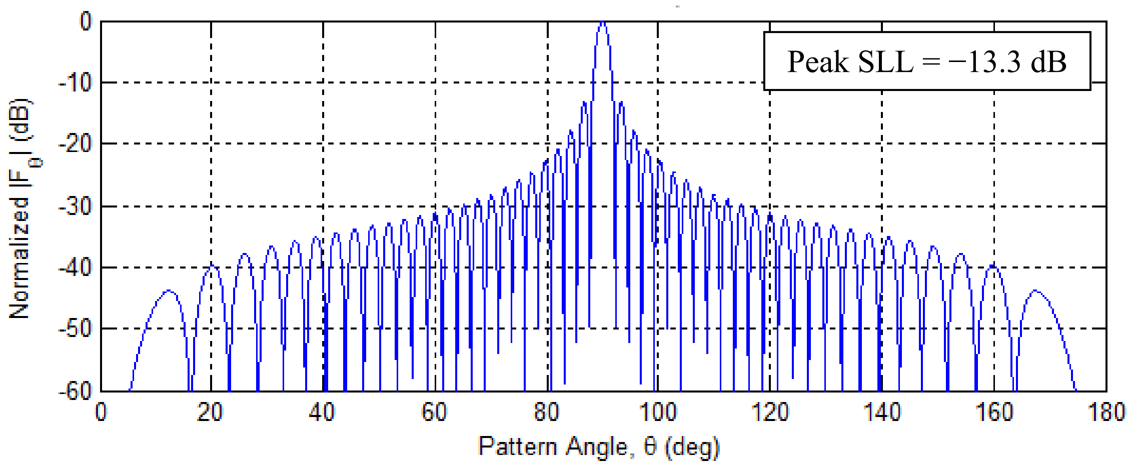


Figure 30. Normalized receive array pattern for the θ component, $\phi = 90°$ cut with configuration settings $N_x = N_z = 10, d_x = d_z = 0.5\lambda, M_x = M_z = 5, l_x = l_z = 5\lambda$.

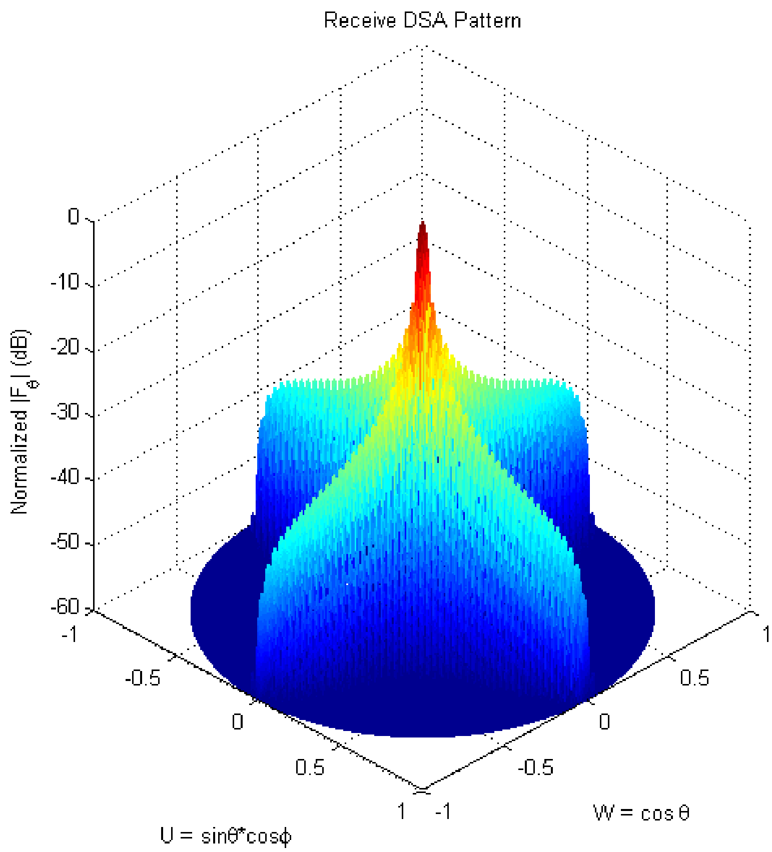


Figure 31. Normalized mesh plot of receive array pattern for the θ component with configuration settings $N_x = N_z = 10, d_x = d_z = 0.5\lambda, M_x = M_z = 5, l_x = l_z = 5\lambda$.

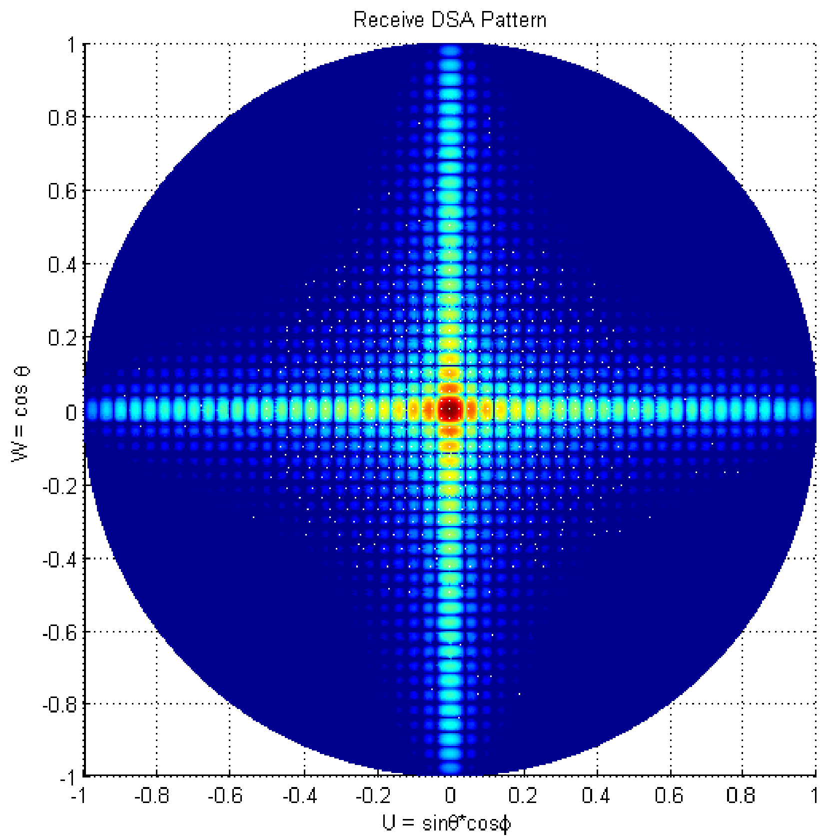


Figure 32. Normalized contour plot of transmit array pattern for the θ component with configuration settings $N_x = N_z = 10, d_x = d_z = 0.5\lambda, M_x = M_z = 5, l_x = l_z = 5\lambda$.

The normalized two-way pattern of the θ component, $\phi = 90°$ pattern cut, for the configuration settings given in Figure 26 is shown in Figure 33. It is observed that the grating lobes at θ pattern angles 11°, 53.2°, 78.7°, 101.3°, 126.8° and 169° of the transmit array have been suppressed. The peak SLL of the two-way pattern is −25.5 dB due to the subarray sidelobes at θ pattern angles 86.7° and 93.3°. The grating lobes at $\theta = 78.7°, 101.3°$ are suppressed to −29 dB. The improvement factor in grating lobe suppression is 25.3 dB. The BWFN of the two-way pattern is approximately 4.6°, and HPBW of the two-way pattern is approximately 1.5°. The three-dimensional mesh plot and contour plot for the pattern angles $0° \le \theta \le 180°$ and $0° \le \phi \le 180°$ of the two-way pattern for the θ component are shown in Figures 34 and 35, respectively. The step size for both θ and ϕ of the pattern plots are set to 0.3°.

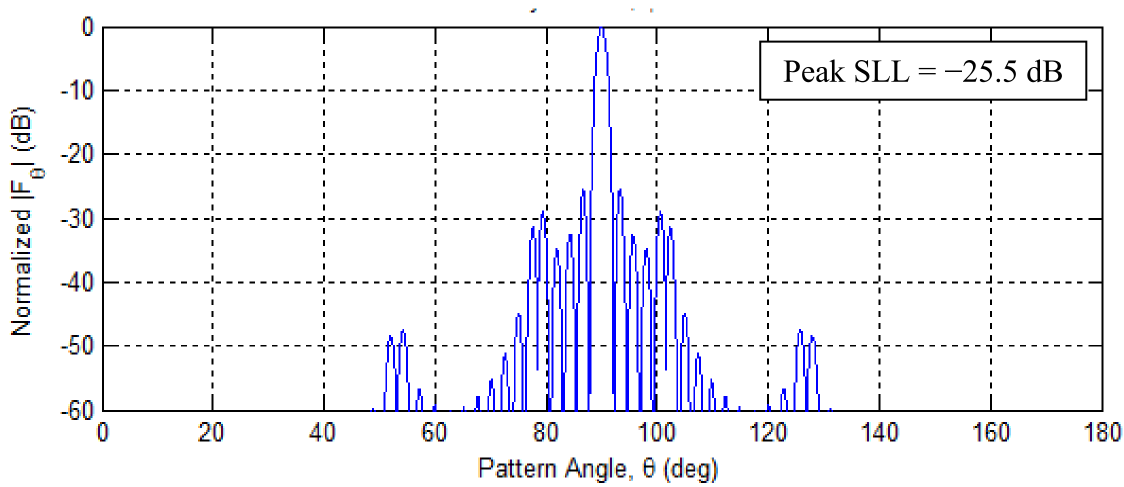


Figure 33. Normalized two-way pattern for the θ component, $\phi = 90°$ cut of a transmit DSA with contiguous receive subarrays configuration.

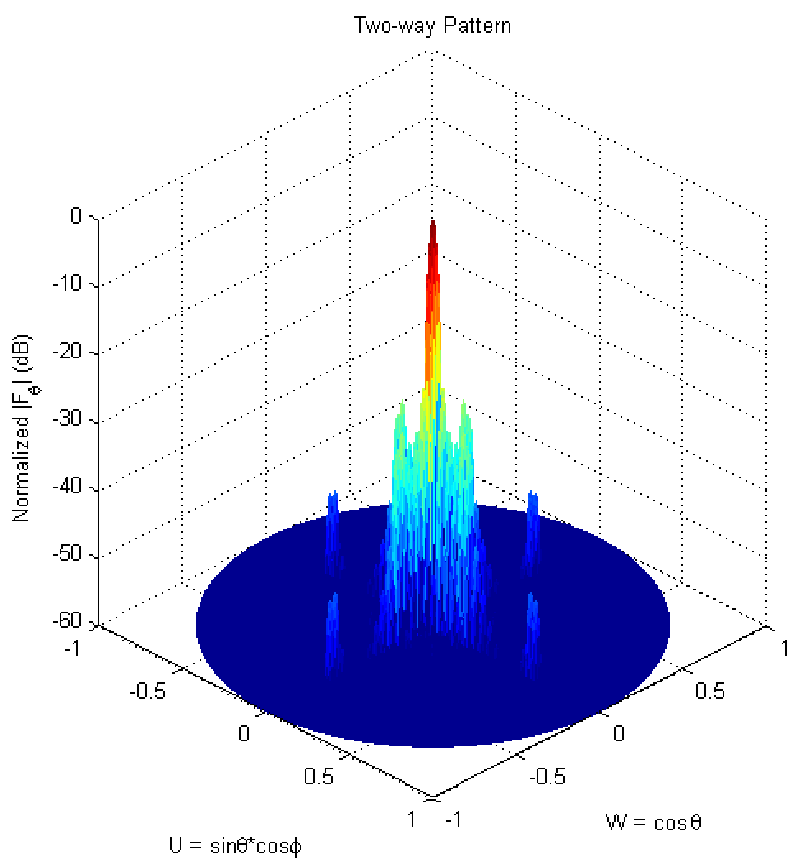


Figure 34. Normalized mesh plot of two-way pattern for the θ component in direction cosine space.

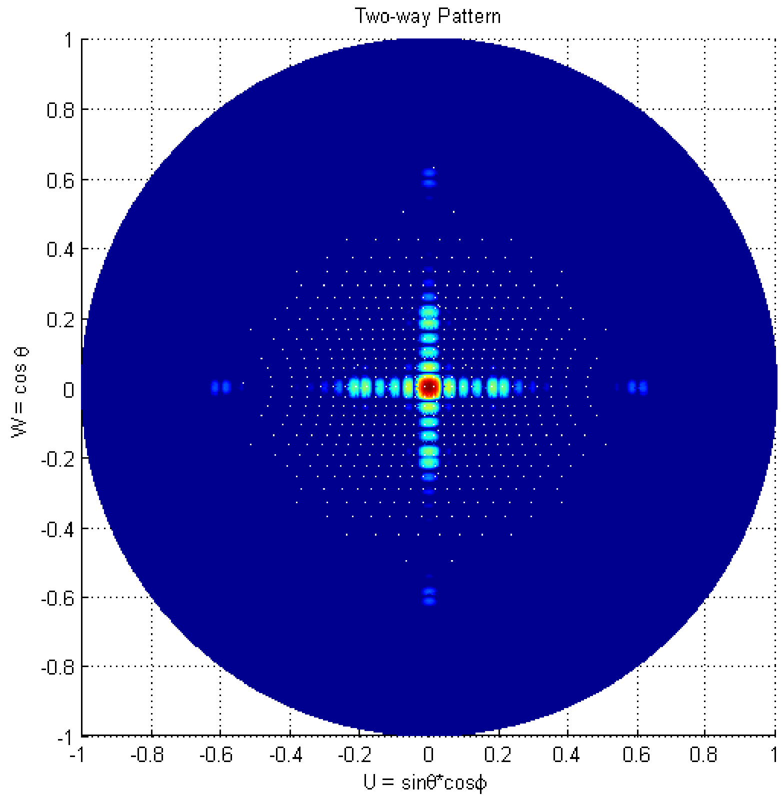


Figure 35. Normalized contour plot of two-way pattern for the θ component in direction cosine space.

Using an integration interval of ten for both θ and ϕ, we get the power gain results of the transmit array, receive array, as well as the two-way power gain shown in Figure 36. The simulation results of the DSA configuration settings in Figure 26 are summarized in Table 1.

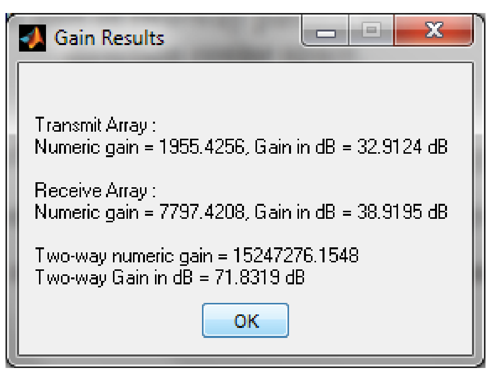


Figure 36. Power gain results of transmit DSA with contiguous receive subarrays configuration.

Table 1. Summary of simulation results of transmit DSA with contiguous receive subarrays configuration.

	Peak SLL	BWFN	HPBW	Power Gain
Transmit Array	−3.7 dB	4.6°	2°	32.9 dB
Receive Array	−13.3 dB	4.6°	2°	38.9 dB
Two-way Pattern	−25.5 dB	4.6°	1.5°	71.8 dB

From the simulation results, it is observed that two-way pattern multiplication approach is effective in suppression of grating lobes. In this example, a grating lobe suppression of at least 25 dB can be achieved.

In the next example, both transmit and receive antennas are configured as widely spaced, thinned arrays [15] to achieve a narrow scanned beam for very good angular resolution without the cost and space required of a fully filled array. Chebyshev illumination is applied to achieve low sidelobe patterns. The transmit array and receive array configuration settings is given in Figure 37.

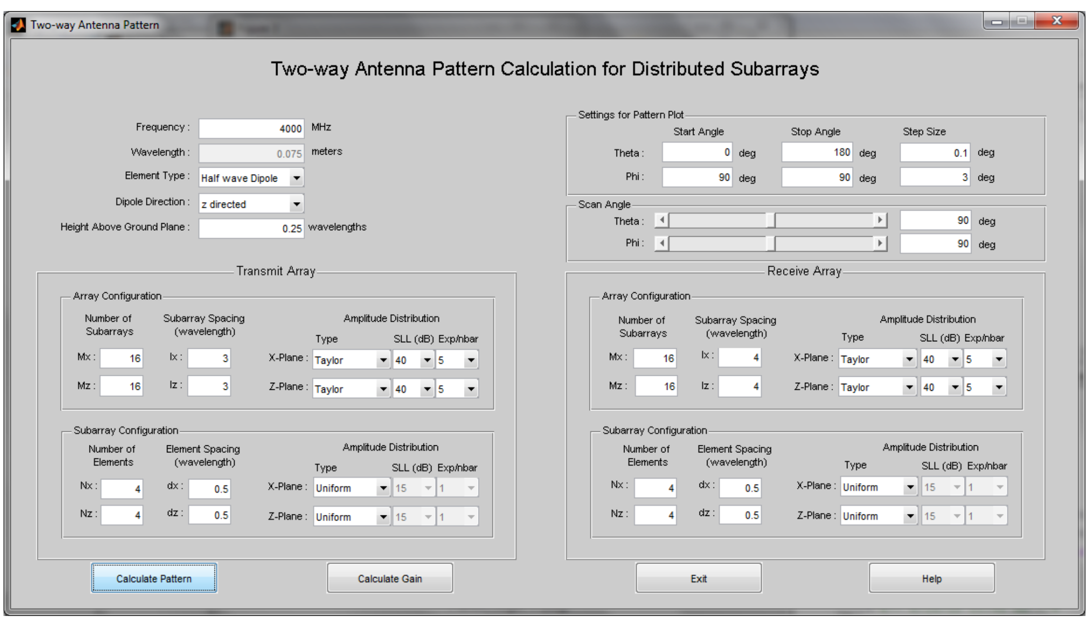


Figure 37. Configuration settings for thinned transmit and receive arrays.

The transmit array is made up of 16×16 subarrays with 3λ spacing. The receive array is made of 16×16 subarrays with 4λ spacing. Each subarray consists of 4×4 *z*-directed half-wave dipole elements placed at a height of 0.25λ above a ground plane. Taylor amplitude distribution with 40 dB SLL and $\bar{n} = 5$ is applied to both transmit and receive array configurations. Beam scan angles $\theta_S = \phi_S = 90°$ are considered. The normalized transmit array pattern of the 16×16 subarrays for the θ component with pattern angle range $0° \le \theta \le 180°$, $\phi = 90°$ cut is shown in Figure 38. The peak SLL due to grating lobes at $\theta = 70.6°, 109.4°$ is observed to be −7.9 dB. The BWFN and HPBW of the transmit array pattern is approximately 4.4° and 1.5°, respectively. The three-dimensional mesh plot and contour plot for the pattern angles $0° \le \theta \le 180°$ and $0° \le \phi \le 180°$ of the transmit array pattern for the θ component are shown in Figures 39 and 40, respectively. A step size of $0.1°$ is used for both θ and ϕ of the pattern plots.

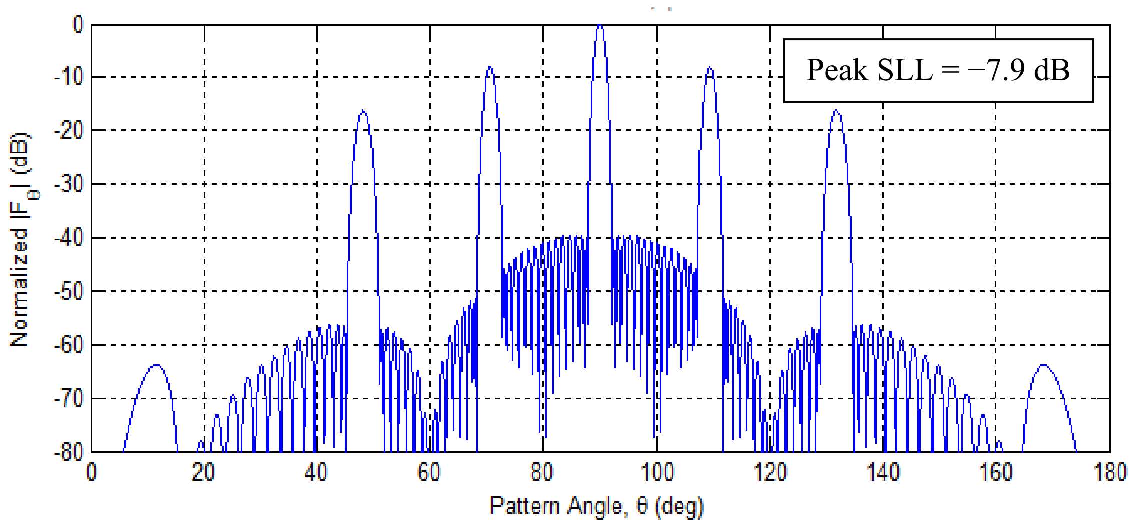


Figure 38. Normalized transmit array pattern of 16×16 subarrays with spacing 3λ for the θ component, $\phi = 90°$ cut.

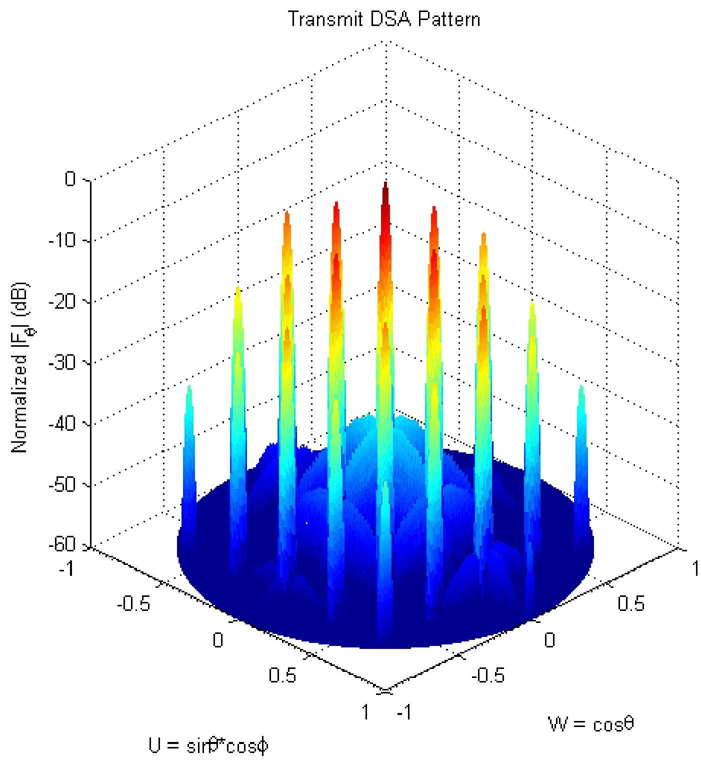


Figure 39. Normalized mesh plot of transmit array pattern of 16×16 subarrays with spacing 3λ for the θ component.

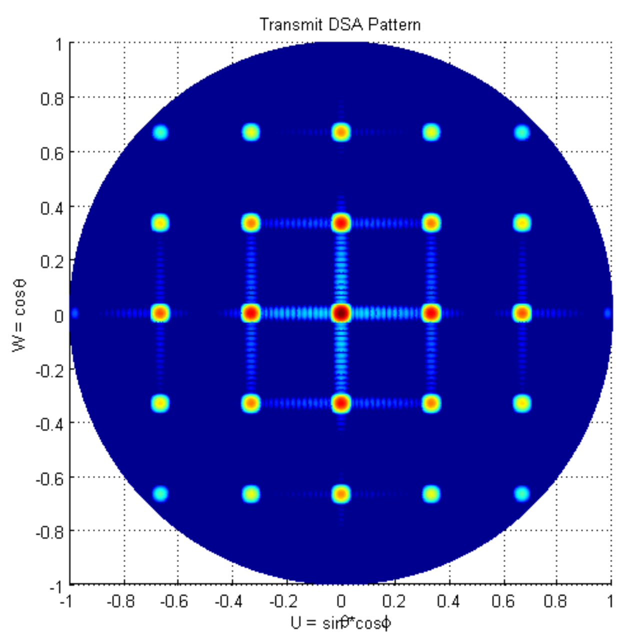


Figure 40. Normalized contour plot of transmit array pattern of 16×16 subarrays with spacing 3λ for the θ component.

The normalized receive array pattern of the 16×16 subarrays for the θ component with pattern angle range $0° \le \theta \le 180°$, $\phi = 90°$ cut is shown in Figure 41. The peak SLL due to grating lobes at $\theta = 75.6°, 104.4°$ is observed to be −4.1 dB. Notice that the grating lobe locations of the transmit and receive arrays are not coincident due to their different subarray spacing. The BWFN and HPBW of the receive array pattern is approximately 3.2° and 1.1°, respectively. The three-dimensional mesh plot and contour plot for the pattern angles $0° \le \theta \le 180°$ and $0° \le \phi \le 180°$ of the receive array pattern for the θ component are shown in Figures 42 and 43, respectively. A step size of 0.1° is used for both θ and ϕ of the pattern plots.

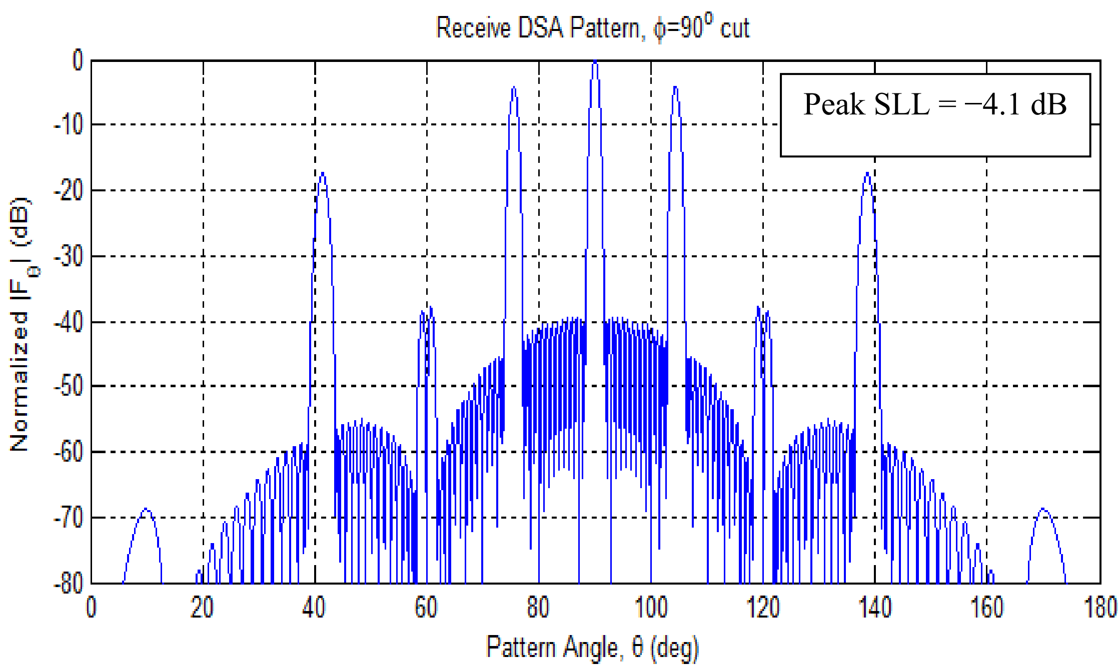


Figure 41. Normalized receive array pattern of 16×16 subarrays with spacing 4λ for the θ component, $\phi = 90°$ cut.

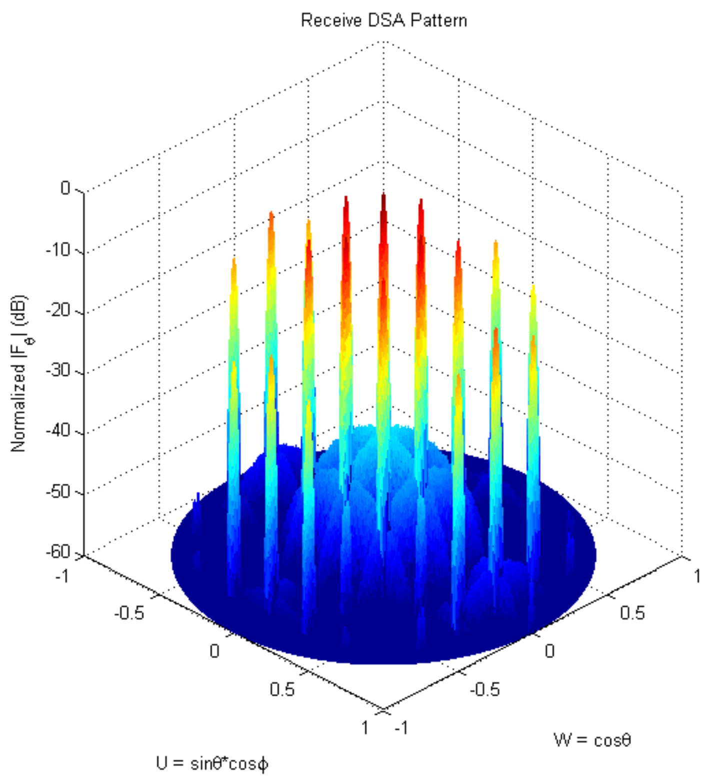


Figure 42. Normalized mesh plot of receive array pattern of 16×16 subarrays with 4λ for the θ component.

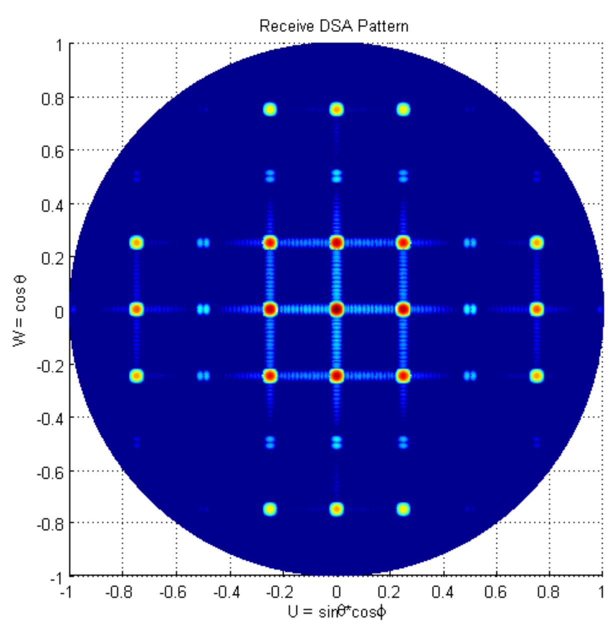


Figure 43. Normalized contour plot of receive array pattern of 16×16 subarrays with 4λ for the θ component.

The normalized two-way pattern of the θ component, $\phi = 90°$ pattern cut, is shown in Figure 44. It is observed that the grating lobes for both transmit and receive arrays have been suppressed since their locations are not coincident, and both transmit and receive array patterns have low SLLs (exclusive of the grating lobes). The peak SLL of the two-way pattern is −49.7 dB at $\theta = 76.1°, 103.9°$. The BWFN of the two-way pattern is approximately 4.4°, and HPBW of the two-way pattern is approximately 0.9°. The three-dimensional mesh plot and contour plot for the pattern angles $0° \leq \theta \leq 180°$ and $0° \leq \phi \leq 180°$ of the two-way pattern for the θ component are shown in Figures 45 and 46, respectively. The step size for both θ and ϕ of the pattern plots are set to 0.1°.

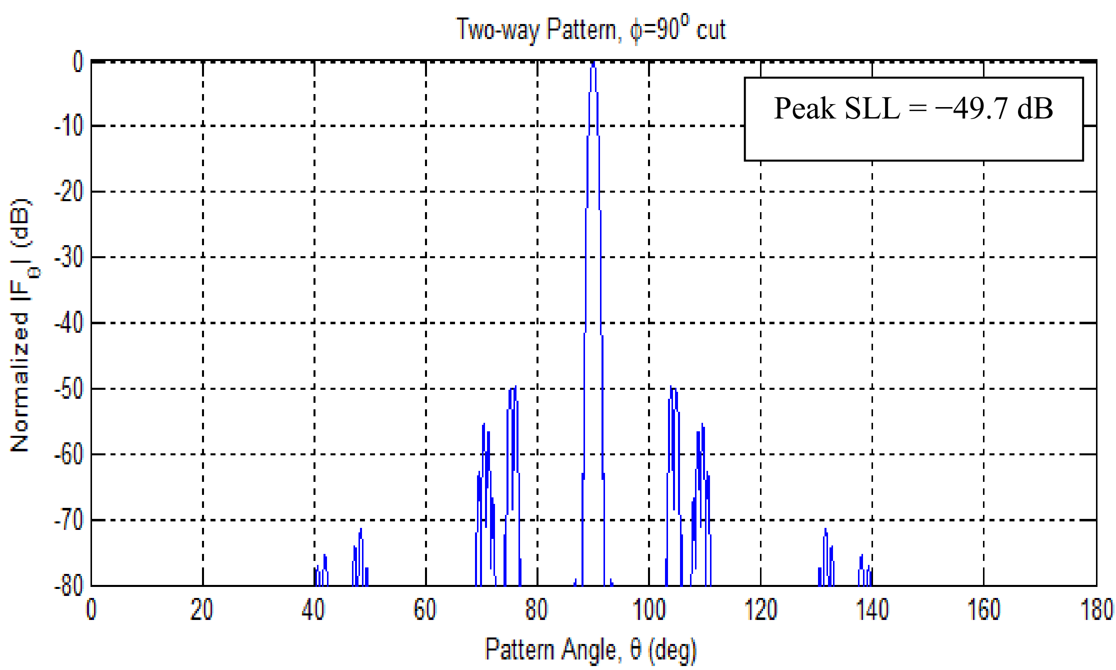


Figure 44. Normalized two-way pattern of thinned transmit and receive arrays for the θ component, $\phi = 90°$ cut.

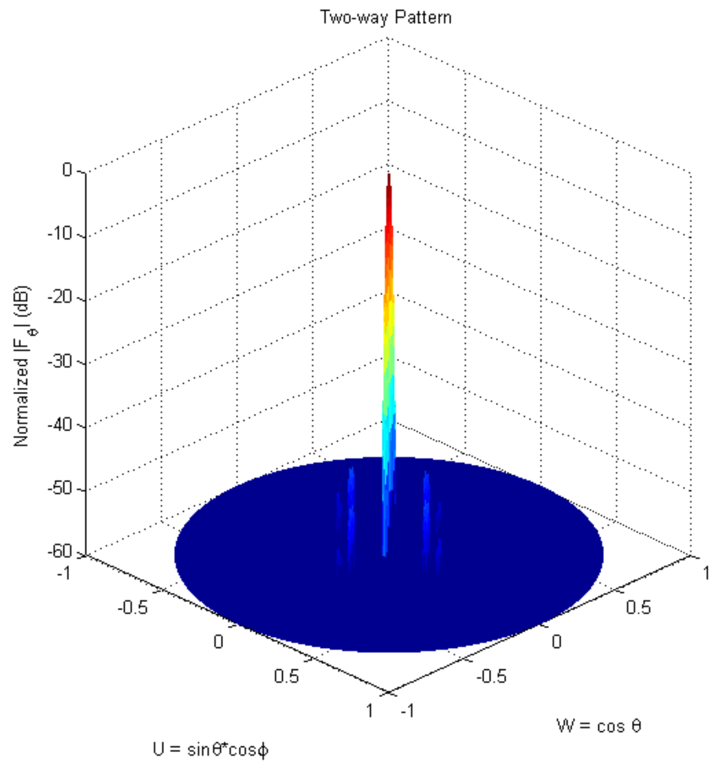


Figure 45. Normalized mesh plot of two-way pattern of thinned transmit and receive arrays for the θ component.

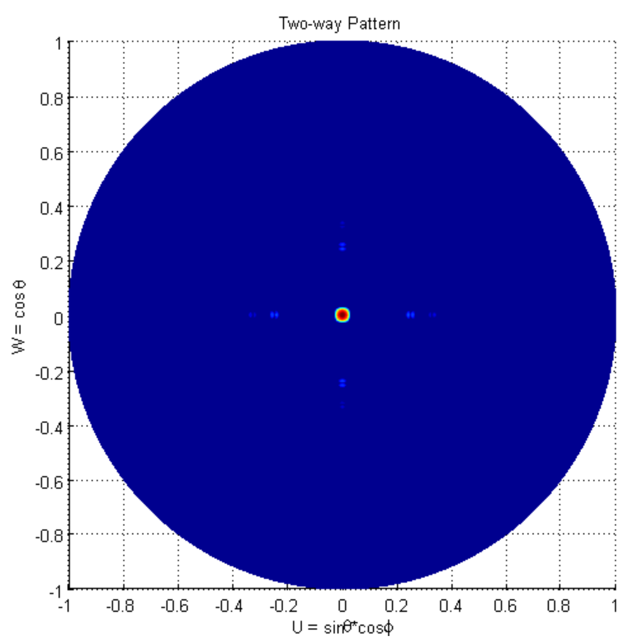


Figure 46. Normalized contour plot of two-way pattern of thinned transmit and receive arrays for the θ component.

Using an integration interval of 18 for both θ and ϕ, we get the power gain results of the transmit array, receive array, as well as the two-way power gain shown in Figure 47. The simulation results are summarized in Table 2.

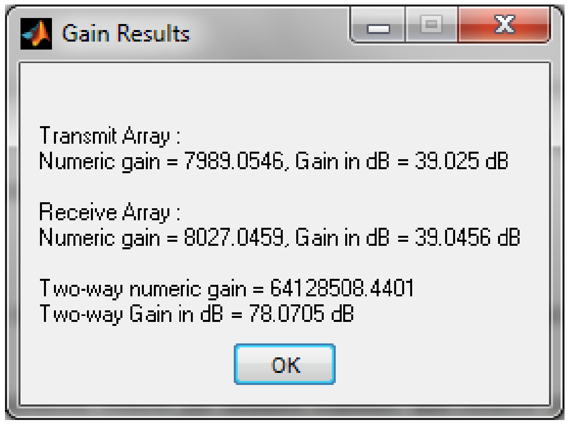


Figure 47. Power gain results using thinned transmit and receive array configurations.

Table 2. Summary of simulation results for thinned transmit and receive array configurations.

	Peak SLL	BWFN	HPBW	Power Gain
Transmit Array	−7.9 dB	4.4°	1.5°	39 dB
Receive Array	−4.1 dB	3.2°	1.1°	39 dB
Two-way Pattern	−49.7 dB	4.4°	0.9°	78 dB

The transmit array pattern, receive array pattern and two-way pattern when the main beam is scanned to $\theta_s = 100°$ are shown in Figures 48, 49 and 50, respectively. It is observed that the peak SLL, BWFN and HPBW are typically unchanged for small beam scan angles.

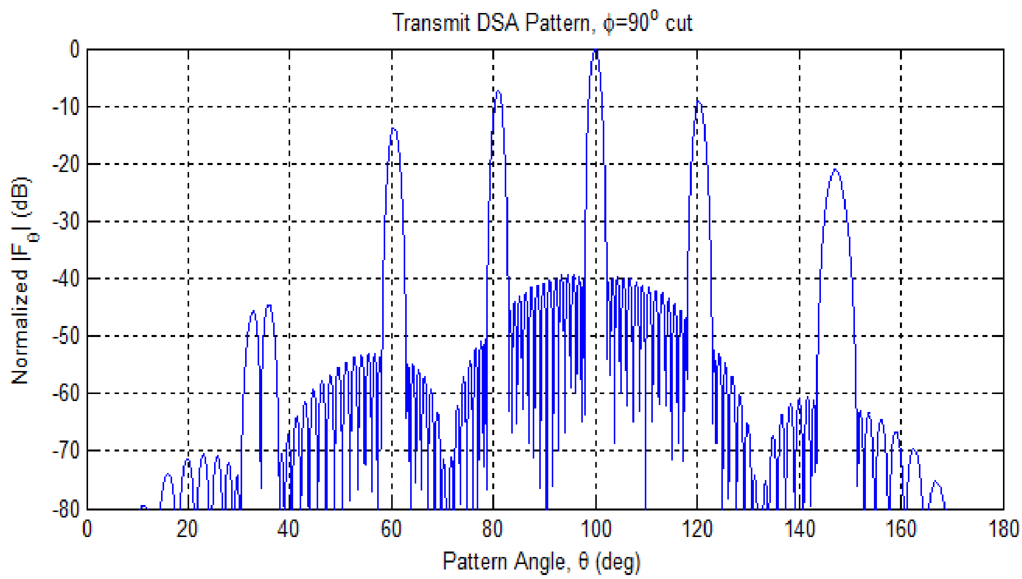


Figure 48. Normalized transmit array pattern of 16×16 subarrays with spacing 3λ when scanned to $\theta_s = 100°$.

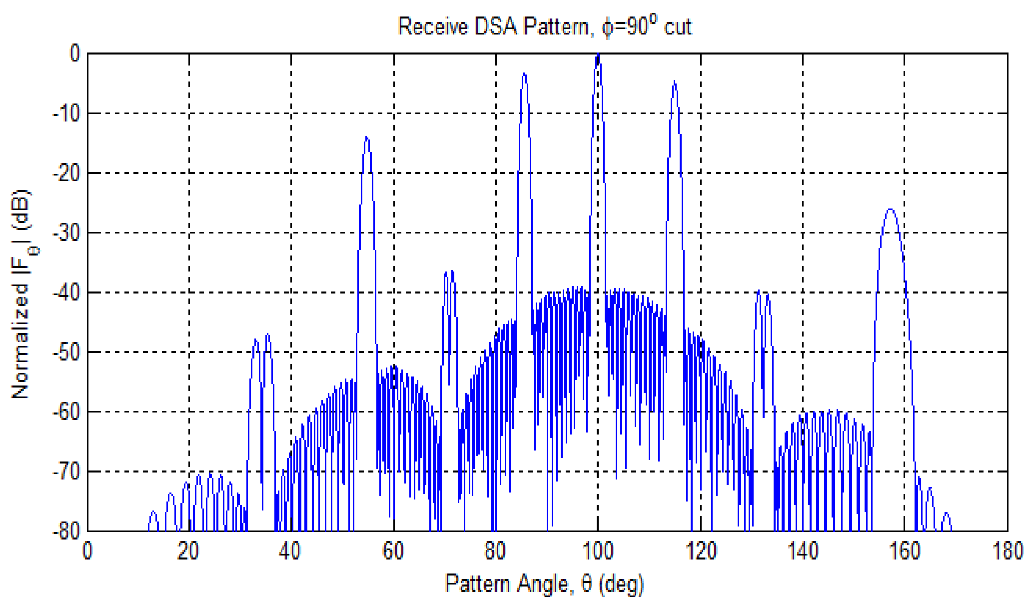


Figure 49. Normalized received array pattern of 16×16 subarrays with spacing 4λ when scanned to $\theta_s = 100°$.

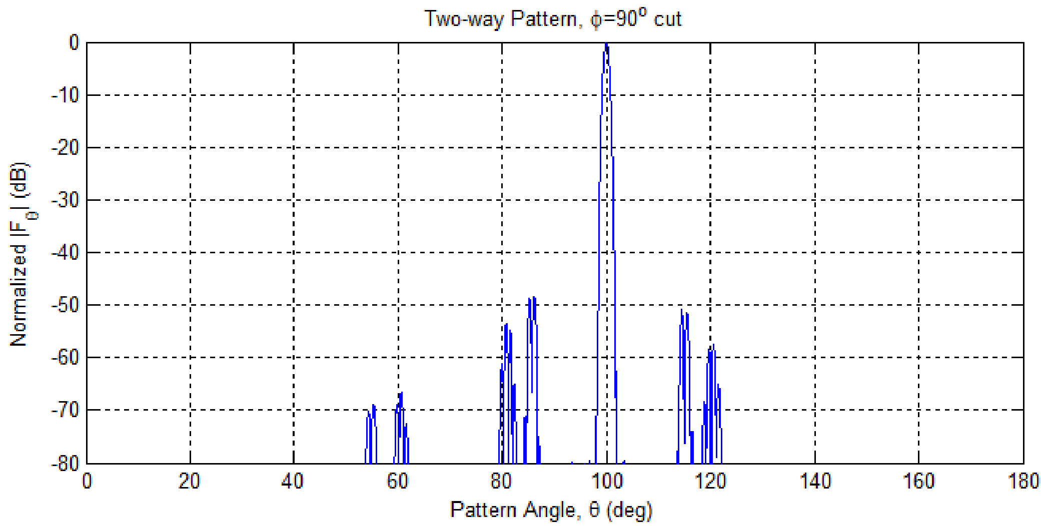


Figure 50. Normalized two-way pattern when scanned to $\theta_s = 100°$.

The transmit array pattern, receive array pattern and two-way pattern when the main beam is scanned to $\theta_s = 140°$ are shown in Figures 51, 52 and 53, respectively. For large scan angles, beam broadening effect is more evident, and the peak SLL is increased. The peak SLL for the two-way pattern is −40.2 dB when the beam is scanned to $\theta_s = 140°$.

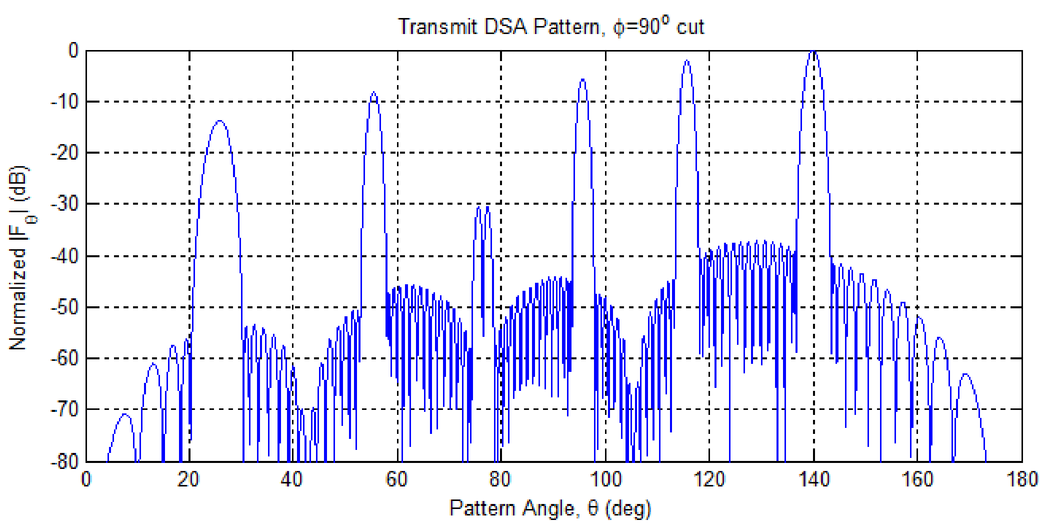


Figure 51. Normalized transmit array pattern of 16×16 subarrays with spacing 3λ when scanned to $\theta_s = 140°$.

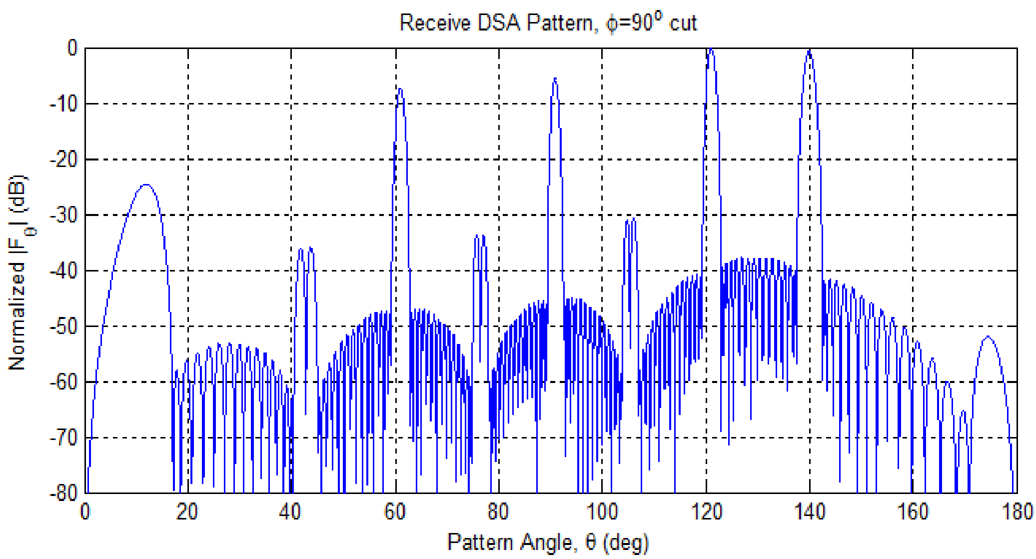


Figure 52. Normalized received array pattern of 16×16 subarrays with spacing 4λ when scanned to $\theta_s = 140°$.

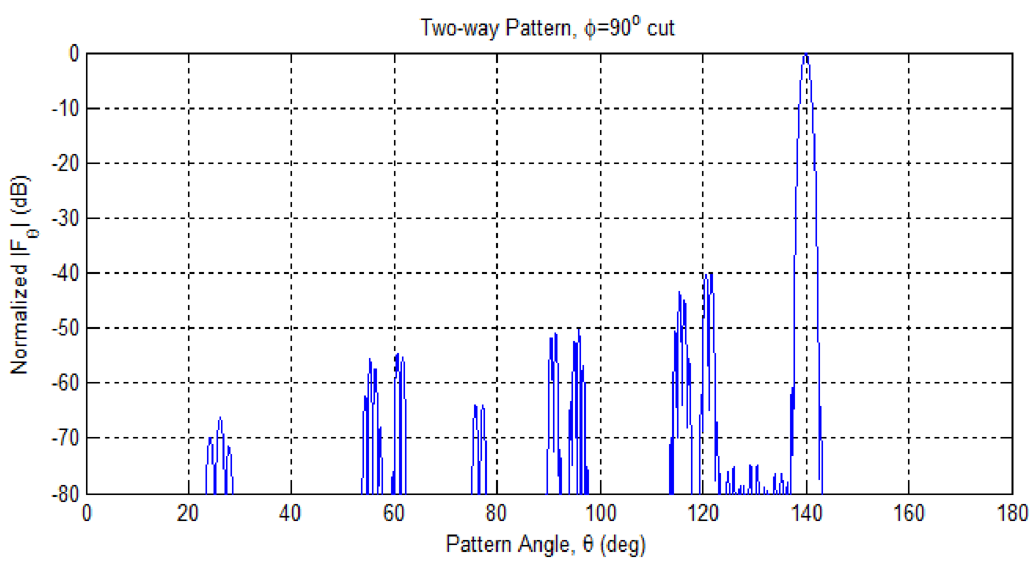


Figure 53. Normalized two-way pattern when scanned to $\theta_s = 140°$.

The two-way patterns of beams scanned at increments of $\Delta\theta_s = 5°$ for $75° \leq \theta_s \leq 105°$ are illustrated in Figure 54.

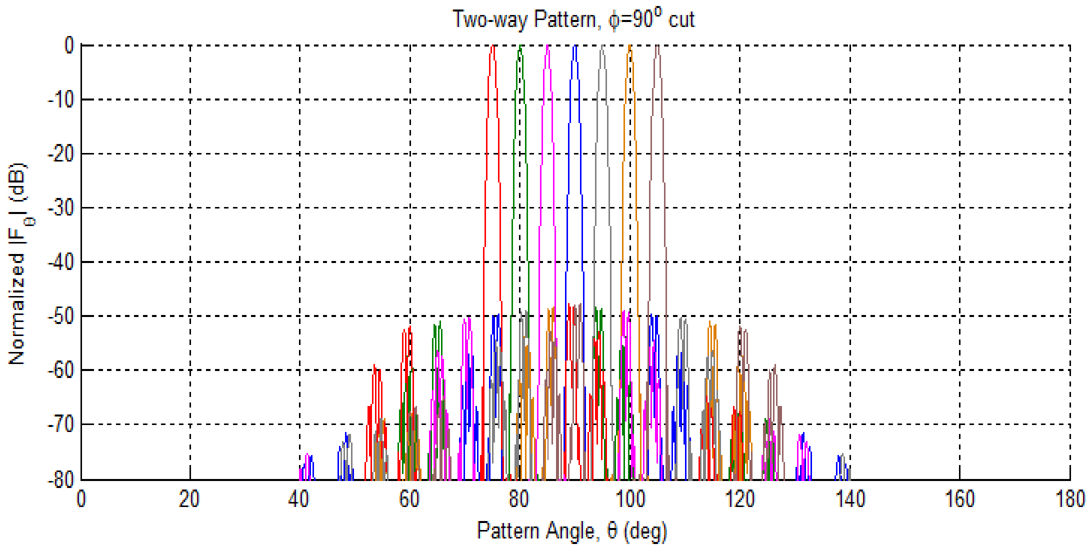


Figure 54. Normalized two-way pattern of multiple beam scanning for $75° \leq \theta_s \leq 105°$ with $\Delta\theta_s = 5°$

We see from the simulation results that using separate transmit and receive array patterns with non-coincident grating lobe locations and low sidelobe illumination, the grating lobes can be effectively suppressed. Although beam broadening usually accompanies low sidelobe illuminations, a narrow beamwidth of the two-way pattern can still be achieved with wide subarray spacing.

C. SUMMARY

A simulation tool was developed and implemented in MATLAB to perform the two-way pattern calculation of transmit and receive DSAs. The features of the simulation tool were described in this chapter. The simulation tool was also able to perform gain calculation for the transmit and receive DSAs, as well as compute the two-way gain.

Using the simulation tool, we demonstrated the effectiveness of the two-way pattern multiplication approach to suppress undesired grating lobes by placement of subarray nulls at the grating lobe locations. In this chapter, we also showed that low SLL and narrow HPBW of the two-way antenna pattern can be achieved with thinned transmit and receive arrays consisting of widely-spaced subarrays with non-coincident grating lobe locations.

THIS PAGE INTENTIONALLY LEFT BLANK

V. SUMMARY AND RECOMMENDATIONS

A. SUMMARY

The primary objective of this research was to develop a simulation tool to investigate the behavior and effectiveness in suppressing undesired grating lobes using the approach of two-way antenna pattern in DSA design. The fundamental array theory and principle of pattern multiplication, which formed the basis of the simulation tool design, were discussed. Using the principle of pattern multiplication, we can suppress grating lobes by placement of subarray nulls at grating lobe locations. A simple program was developed in MATLAB to allow the user to visualize the placement of grating lobes and nulls in direction cosine space for a DSA configuration in the visible region. A simulation tool with GUI was developed and implemented in MATLAB to perform the two-way antenna pattern and power gain calculations for user configured DSAs. The program is capable of performing two-way pattern and power gain calculations for linear or planar DSAs consisting of isotropic elements, half-wave dipoles or short dipoles above a ground plane. The program is able to present the simulation results in the θ pattern cut, ϕ pattern cut, or as a three-dimensional mesh plot in direction cosine space. The program GUI provides a convenient way for the user to tweak the design configurations very quickly by changing the DSA parameters. The program can serve as a useful tool for both students and electromagnetic professionals to determine and study the two-way pattern and power gain of different transmit/receive DSA designs.

The effectiveness of the two-way pattern multiplication approach to suppress undesired grating lobes by placement of subarray nulls at the grating lobe locations was demonstrated using the simulation tool. For a transmit DSA with subarray spacing of 5λ and uniform amplitude illumination, a SLL of −25.5 dB was achieved. Using the simulation tool, we demonstrated that low SLL and narrow HPBW of the two-way antenna pattern can be achieved using thinned transmit and receive arrays consisting of widely-spaced subarrays with non-coincident grating lobe locations. The simulation results showed that a SLL of −49.7 dB and HPBW of 0.9° were obtained for broadside illumination. Simulation was carried out to examine the effects on the two-way pattern when different scan angles were applied. The simulation results showed that beam broadening was evident and peak SLL was increased for the two-way antenna pattern when large scan angles were applied to the main beams of the transmit and receive arrays. It was assumed that the main beams of the transmit and receive arrays were scanned to the same angle.

B. RECOMMENDATIONS

At this time, the simulation tool allows us to study the two-way pattern for equally-spaced linear or rectangular DSAs. Other common array configurations include triangular arrays and circular arrays. A triangular grid is known to be more efficient than a rectangular grid for the suppression of grating lobes because fewer elements are required for a given aperture size [10]. Circular arrays, in which the elements are placed in a circular ring, have the advantage of symmetry in azimuth, making them suitable for applications that require full 360° coverage, such as direction-finding, air and space navigation, and underground propagation [16]. It would be useful to upgrade the simulation tool to be able to compute the two-way pattern of triangular, circular and other array configurations to allow the study of their two-way patterns.

Phase shifters are used to control the location and shape of the antenna beam. In practice, most phase shifters are digitally controlled devices, which allow only discrete values for the phase shift. Therefore, truncation or round-off error must be introduced onto the phase shifts, and this yields a periodic quantization error that gives rise to quantization lobes [10]. The current version of the simulation tool assumes continuous phase shift in the calculation and does not account for the effect of phase quantization. It is essential to include some round-off error or truncation methods for the phase shift computation as a future improvement to the program.

APPENDIX A. DIPOLES OF ARBITRARY ORIENTATION[1]

The dipole is centered at the origin as shown in Figure A1. The dipole direction is given by the vector

$$\hat{I} = \hat{x}\underbrace{\sin\theta_a \cos\phi_a}_{u_a} + \hat{y}\underbrace{\sin\theta_a \sin\phi_a}_{v_a} + \hat{z}\underbrace{\cos\theta_a}_{w_a} . \tag{43}$$

The dipole direction cosines are

$$\begin{aligned} &x\text{-direction cosine}: u_a = \cos\alpha_x = \sin\theta_a \cos\phi_a \\ &y\text{-direction cosine}: v_a = \cos\alpha_y = \sin\theta_a \sin\phi_a \\ &z\text{-direction cosine}: w_a = \cos\alpha_z = \cos\theta_a . \end{aligned} \tag{44}$$

The length of the dipole is L, and for a half-wave dipole the current distribution is given by the function of the path length variable ℓ'

$$I(\ell') = I_m \cos(\beta\ell'), \quad -L/2 \le \ell' \le L/2 \tag{45}$$

where I_m is the maximum value at the feed point. For an ideal (Hertzian) dipole with constant current

$$I(\ell') = I_m, \quad -L/2 \le \ell' \le L/2, \quad L << \lambda . \tag{46}$$

The direction cosines for the far-field observation point are

$$\begin{aligned} &x\text{-direction cosine}: u = \sin\theta\cos\phi \\ &y\text{-direction cosine}: v = \sin\theta\sin\phi \\ &z\text{-direction cosine}: w = \cos\theta . \end{aligned} \tag{47}$$

[1] The material for Appendix A is from “Arbitrary dipoles” unpublished notes of David C. Jenn.

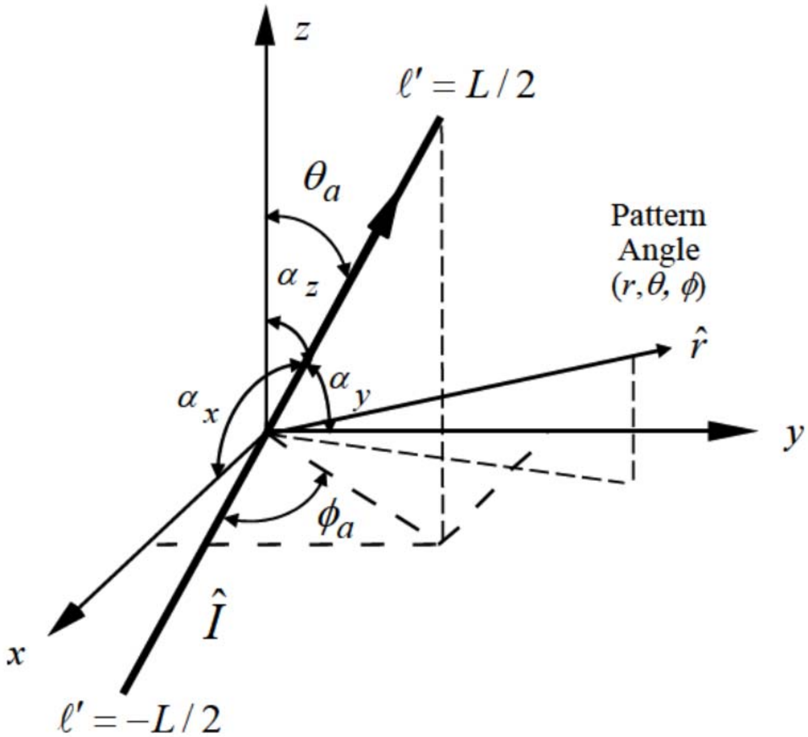


Figure A1. Dipole of arbitrary orientation centered at the origin

The far-field radiation integral can be written as

$$\vec{E}(\theta,\phi) = \frac{-j\beta\eta_o\hat{I}}{4\pi r}e^{-j\beta r}\int_{-L/2}^{L/2} I(\ell')e^{j\beta\ell'(\hat{r}\bullet\hat{I})}d\ell' \tag{48}$$

where it is understood that the $\hat{r}$ component is discarded. For the ideal dipole the integral is

$$\int_{-L/2}^{L/2} I(\ell')e^{j\beta\ell'(\hat{r}\bullet\hat{I})}d\ell' = I_m L\,\mathrm{sinc}\left(\frac{\beta L\hat{r}\bullet\hat{I}}{2}\right) \approx I_m L \quad (L \ll \lambda), \tag{49}$$

and the resulting spherical field components are

$$\begin{aligned} E_\theta &= \vec{E}(\theta,\phi)\bullet\hat{\theta} = \frac{-j\beta\eta_o I_m L}{4\pi r}e^{-j\beta r}\hat{I}\bullet\hat{\theta} \\ E_\phi &= \vec{E}(\theta,\phi)\bullet\hat{\phi} = \frac{-j\beta\eta_o I_m L}{4\pi r}e^{-j\beta r}\hat{I}\bullet\hat{\phi}. \end{aligned} \tag{50}$$

From the transform tables:

$$\begin{aligned}\hat{\theta} &= \hat{x}\cos\theta\cos\phi + \hat{y}\cos\theta\sin\phi - \hat{z}\sin\theta \\ \hat{\phi} &= -\hat{x}\sin\phi + \hat{y}\cos\phi\,.\end{aligned} \tag{51}$$

For a Hertzian dipole along the *z*-axis $\left(\theta_a = \phi_a = 0°, \hat{I} = \hat{z}\right)$:

$$\begin{aligned}\hat{I} \bullet \hat{\theta} &= \hat{z} \bullet \hat{\theta} = -\sin\theta \\ \hat{I} \bullet \hat{\phi} &= \hat{z} \bullet \hat{\phi} = 0\,,\end{aligned} \tag{52}$$

and the fields reduce to the familiar results

$$\begin{aligned}E_\theta &= \frac{j\beta\eta_o I_m L}{4\pi r} e^{-j\beta r}\sin\theta \\ E_\phi &= 0\,.\end{aligned} \tag{53}$$

For a half-wave dipole $(L = \lambda / 2)$, the integral from (48) is

$$\int_{-L/2}^{L/2} I_m \cos(\beta\ell') e^{j\beta\ell'(\hat{r}\bullet\hat{I})} d\ell' = \left\{ \frac{\mathrm{I}_m e^{j\beta\ell'(\hat{r}\bullet\hat{I})}\left[j\beta\ell'(\hat{r}\bullet\hat{I})\cos(\beta\ell') + \beta\sin(\beta\ell')\right]}{\left[\beta\ell'(\hat{r}\bullet\hat{I})\right]^2 + \beta^2} \right\}_{-L/2}^{L/2}. \tag{54}$$

After applying some trigonometric identities, we get the final result for the far electric field

$$\vec{E}(\theta,\phi) = -\frac{j\eta_o I_m \hat{I}\, e^{-j\beta r}}{2\pi r} \frac{\cos\left(\frac{\pi}{2}\hat{r} \bullet \hat{I}\right)}{1-\left(\hat{r} \bullet \hat{I}\right)^2}. \tag{55}$$

For a half-wave dipole along the z-axis $\left(\theta_a = \phi_a = 0°, \hat{I} = \hat{z}\right)$:

$$\begin{aligned}\hat{r} \bullet \hat{I} &= \cos\theta = \omega\,, \\ \hat{\theta} \bullet \hat{I} &= -\sin\theta\,,\end{aligned} \tag{56}$$

$$\begin{aligned}E_\theta &= \frac{j\eta_o I_m\, e^{-j\beta r}}{2\pi r}\left[\frac{\cos\left(\frac{\pi}{2}\cos\theta\right)}{\sin\theta}\right], \\ E_\phi &= 0\,.\end{aligned} \tag{57}$$

For a half-wave dipole along the *y*-axis $\left(\theta_a = \phi_a = 90°, \hat{I} = \hat{y}\right)$:

$$
\begin{aligned}
&\hat{r} \bullet \hat{I} = \sin\theta \sin\phi = v \,, \\
&\hat{\theta} \bullet \hat{I} = \cos\theta \sin\phi \,, \\
&\hat{\phi} \bullet \hat{I} = \cos\phi \,,
\end{aligned}
\tag{58}
$$

$$
\begin{aligned}
E_\theta &= \frac{j\eta_o I_m \, e^{-j\beta r}}{2\pi r} \left[\frac{\cos\left(\frac{\pi}{2}\sin\theta\sin\phi\right)}{1-\sin^2\theta\sin^2\phi} \right] \cos\theta\sin\phi \,, \\
E_\phi &= \frac{j\eta_o I_m \, e^{-j\beta r}}{2\pi r} \left[\frac{\cos\left(\frac{\pi}{2}\sin\theta\sin\phi\right)}{1-\sin^2\theta\sin^2\phi} \right] \cos\phi \,.
\end{aligned}
\tag{59}
$$

For a half-wave dipole along the *x*-axis $\left(\theta_a = 90°, \phi_a = 0°, \hat{I} = \hat{x}\right)$:

$$
\begin{aligned}
&\hat{r} \bullet \hat{I} = \sin\theta \cos\phi = u \,, \\
&\hat{\theta} \bullet \hat{I} = \cos\theta \cos\phi \,, \\
&\hat{\phi} \bullet \hat{I} = -\sin\phi \,,
\end{aligned}
\tag{60}
$$

$$
\begin{aligned}
E_\theta &= \frac{j\eta_o I_m \, e^{-j\beta r}}{2\pi r} \left[\frac{\cos\left(\frac{\pi}{2}\sin\theta\cos\phi\right)}{1-\sin^2\theta\cos^2\phi} \right] \cos\theta\cos\phi \,, \\
E_\phi &= -\frac{j\eta_o I_m \, e^{-j\beta r}}{2\pi r} \left[\frac{\cos\left(\frac{\pi}{2}\sin\theta\cos\phi\right)}{1-\sin^2\theta\cos^2\phi} \right] \sin\phi \,.
\end{aligned}
\tag{61}
$$

APPENDIX B. MATLAB CODE FOR PLOTTING GRATING LOBES AND NULL LOCATIONS OF ARRAY PATTERN

```
% Plot_rect_nulls_and_grating_xyplane.m
% This code computes and plots the nulls and grating lobes locations
% for rectangular lattice of DSAs in x-y plane in the direction
% cosine space. Comparison is made between XMTR and RCVR DSAs to
% observe the placement of nulls at grating lobe locations to
% suppress grating lobes.
% Modified by Cher Hock Hin
% Date: 8 Aug 2012

clear
rad=pi/180;

%XMTR array settings
dxt=0.5; dyt=0.5; %element spacing in x and y plane (in wavelengths)
Nxt=5; Nyt=5; %number of elements in x and y plane
lxt=5; lyt=5; %subarray spacing in x and y plane (in wavelengths)
%RCVR array settings
dxr=0.5; dyr=0.5; %element spacing in x and y plane (in wavelengths)
Nxr=10; Nyr=10; %number of elements in x and y plane
lxr=5; lyr=5; %subarray spacing in x and y plane (in wavelengths)

%scan angle
thetas=0; thetar=thetas*rad;
phis=0; phir=phis*rad;
us=sin(thetar)*cos(phir); vs=sin(thetar)*sin(phir);
fmhz=100;
f=fmhz*1e6;
wave=3e8/f;
wdx=1/dxt; %lambda/spacing
wdz=1/dyt; %lambda/spacing
N=20; %sets the number of grating lobes to plot
pt=0; lt=0; pr=0; lr=0;
for n=-N:N;
    for m=-N:N
            Dxt=n/dxt/Nxt+us; Dyt=m/dyt/Nyt+vs;
            xlobet=n/lxt+us; ylobet=m/lyt+vs;
            lobet=sqrt(xlobet^2+ylobet^2);
            nullt=sqrt(Dxt^2+Dyt^2);
            if nullt<=1 %find XMTR array nulls
                pt=pt+1;
                Xt(pt)=Dxt; Yt(pt)=Dyt;
            end
            if lobet<=1 %find XMTR array grating lobes
                lt=lt+1;
                Xlobet(lt)=xlobet; Ylobet(lt)=ylobet;
            end

            Dxr=n/dxr/Nxr+us; Dyr=m/dyr/Nyr+vs;
            xlober=n/lxr+us; ylober=m/lyr+vs;
            lober=sqrt(xlober^2+ylober^2);
            nullr=sqrt(Dxr^2+Dyr^2);
            if nullr<=1 %find RCVR array nulls
                pr=pr+1;
                Xr(pr)=Dxr; Yr(pr)=Dyr;
            end
            if lober<=1 %find RCVR array grating lobes
```

```
                lr=lr+1;
                Xlober(lr)=xlober; Ylober(lr)=ylober;
            end
            %main beam
            if m==0 & n==0, X0t=Dxt; Y0t=Dyt; end
    end
end
%unit circle
t=0:360; cx=cos(t*rad); cy=sin(t*rad);

figure(1), clf %plot for XMTR array
plot(Xt,Yt,'bd',Xlobet,Ylobet,'m+',X0t,Y0t,'r*',cx,cy,'-r')
legend('Nulls','Grating Lobes','Main Beam','Location','EastOutside')
axis([-1,1,-1,1])
title({['XMTR Array, \theta_s= ',num2str(thetas),'^o, \phi_s=
',num2str(phis),'^o'];...
    ['N_x = ',num2str(Nxt),', d_x = ',num2str(dxt),'\lambda, ','l_x =
',num2str(lxt),'\lambda, ',...
    'N_y = ',num2str(Nyt),', d_y = ',num2str(dyt),'\lambda, ','l_y =
',num2str(lyt),'\lambda'];})
axis square
xlabel('u=sin\theta cos\phi')
ylabel('v=sin\theta sin\phi')

figure(2), clf %plot for RCVR array
plot(Xr,Yr,'ks',Xlober,Ylober,'bx',X0t,Y0t,'r*',cx,cy,'-r')
legend('Nulls','Grating Lobes','Main Beam','Location','EastOutside')
axis([-1,1,-1,1])
title({['RCVR Array, \theta_s= ',num2str(thetas),'^o, \phi_s=
',num2str(phis),'^o'];...
    ['N_x = ',num2str(Nxr),', d_x = ',num2str(dxr),'\lambda, ','l_x =
',num2str(lxr),'\lambda, ',...
    'N_y = ',num2str(Nyr),', d_y = ',num2str(dyr),'\lambda, ','l_y =
',num2str(lyr),'\lambda'];})
axis square
xlabel('u=sin\theta cos\phi')
ylabel('v=sin\theta sin\phi')

figure(3), clf %plot overlap of XMTR and RCVR array
plot(Xt,Yt,'bd',Xlobet,Ylobet,'m+',Xr,Yr,'ks',Xlober,Ylober,'bx',...
     X0t,Y0t,'r*',cx,cy,'-r')
legend('XMTR Nulls','XMTR Grating Lobes','RCVR Nulls','RCVR Grating
Lobes','Main Beam','Location','EastOutside')
axis([-1,1,-1,1])
title(['Overlap of Grating Lobes & Nulls for XMTR & RCVR Arrays, \theta_s=
',num2str(thetas),'^o, \phi_s= ',num2str(phis),'^o'])
axis square
xlabel('u=sin\theta cos\phi')
ylabel('v=sin\theta sin\phi')
```

APPENDIX C. MATLAB CODE FOR SIMULATION TOOL

```
function dsaplot(action)
% dsaplot.m
% Version: 2.1
% Author: Cher Hock Hin
% Advisor: Professor David C. Jenn
% Date: 13 August 2012
%
% Uses the function "caf_hdip2.m", "caf_sdip2.m" and "caf_iso2.m
% to compute XMTR and RCVR array patterns for half wave dipoles,
% short dipoles and isotropic elements which return outputs in
% complex form (non-dB).
% Two-way pattern is then computed using pattern multiplication.
% Two-way gain is also computed using method of numerical
% integration.
% Pattern and gain calculation is based on user configuration
% of XMTR and RCVR planar arrays consisting of dipoles or
% isotropic elements above a ground plane. The arrays are
% oriented in the x-z plane; ground plane is the x-z plane;
% y axis is normal to ground plane.

switch(action)

    %Case for frequency
    case 'frequency'
      h_frequency = findobj(gcf,'Tag','frequency');
      freq_str = get(h_frequency,'String');
      freq = getFreq(freq_str);
      set(h_frequency,'String',num2str(freq));
      wavelength = 3e8/freq/1e6;
      h_wavelength = findobj(gcf,'Tag','wavelength');
      set(h_wavelength,'String',num2str(wavelength));

    %Case for element type selection
    case 'eltype'
      h_eltype = get(findobj(gcf,'Tag','eltype'),'Value');
      if h_eltype == 3
          set(findobj(gcf,'Tag','dipdir'),'Enable','off');
      else
          set(findobj(gcf,'Tag','dipdir'),'Enable','on');
      end

    %Case for phi start angle
    case 'pstart'
    h_pstart = findobj(gcf,'Tag','pstart');
    pstart_str = get(h_pstart,'String');
    pstart = getPStart(pstart_str);
    set(h_pstart,'String',num2str(pstart));

    %Case for phi stop angle
    case 'pstop'
    h_pstop = findobj(gcf,'Tag','pstop');
    pstop_str = get(h_pstop,'String');
    pstop = getPStop(pstop_str);
    set(h_pstop,'String',num2str(pstop));

    %Case for theta start angle
    case 'tstart'
```

```
    h_tstart = findobj(gcf,'Tag','tstart');
    tstart_str = get(h_tstart,'String');
    tstart = getTStart(tstart_str);
    set(h_tstart,'String',num2str(tstart));

    %Case for theta stop angle
    case 'tstop'
    h_tstop = findobj(gcf,'Tag','tstop');
    tstop_str = get(h_tstop,'String');
    tstop = getTStop(tstop_str);
    set(h_tstop,'String',num2str(tstop));

    %Case for thetas_slider
    case 'thetas_slider'
      h_thetas_slider = findobj(gcf,'Tag','thetas_slider');
      thetas = ceil(get(h_thetas_slider,'Value'));
      h_thetas = findobj(gcf,'Tag','thetas');
      set(h_thetas,'String',num2str(thetas));

    case 'thetas'
      h_thetas = findobj(gcf,'Tag','thetas');
      thetas = str2num(get(h_thetas,'String'));
      thetas = getScanAngle(thetas); %validate range between 0 to 180
      h_thetas_slider = findobj(gcf,'Tag','thetas_slider');
      set(h_thetas_slider,'Value',ceil(thetas));
      set(h_thetas,'String',num2str(thetas));

    %Case for phis_slider
    case 'phis_slider'
      h_phis_slider = findobj(gcf,'Tag','phis_slider');
      phis = ceil(get(h_phis_slider,'Value'));
      h_phis = findobj(gcf,'Tag','phis');
      set(h_phis,'String',num2str(phis));

    case 'phis'
      h_phis = findobj(gcf,'Tag','phis');
      phis = str2num(get(h_phis,'String'));
      phis = getScanAngle(phis); %validate range between 0 to 180
      h_phis_slider = findobj(gcf,'Tag','phis_slider');
      set(h_phis_slider,'Value',ceil(phis));
      set(h_phis,'String',num2str(phis));

    %Case for theta step size
    case 'delt'
      h_delt = get(findobj(gcf,'Tag','delt'),'Value');
      if h_delt >= 3, warndlg('Setting step size to < 1 degree will greatly
increase computation time!','Warning!') ; end

    %Case for phi step size
    case 'delp'
      h_delp = get(findobj(gcf,'Tag','delp'),'Value');
      if h_delp >= 3, warndlg('Setting step size to < 1 degree will greatly
increase computation time!','Warning!') ; end

    %Case for Transmit Array parameters
    case 'Nxt'
      h_Nxt = findobj(gcf,'Tag','Nxt');
      nxt_str = get(h_Nxt,'String');
      Nxt = getNEL(nxt_str);
      set(h_Nxt,'String',num2str(Nxt));
      if Nxt == 1
         dxt = 0;
```

```
        set(findobj(gcf,'Tag','dxt'),'String',num2str(dxt));
        warndlg('For a single element, array spacing is set to zero.',...
                 '  Array Spacing','help');
     elseif Nxt > 1
       dxt = str2num(get(findobj(gcf,'Tag','dxt'),'String'));
        if dxt == 0
           set(findobj(gcf,'Tag','dxt'),'String',num2str(1));
           warndlg('For the number of elements, array spacing cannot be
zero.',...
              '  Array Spacing','help');
        end
     end

  case 'Nzt'
     h_Nzt = findobj(gcf,'Tag','Nzt');
     nzt_str = get(h_Nzt,'String');
     Nzt = getNEL(nzt_str);
     set(h_Nzt,'String',num2str(Nzt));
     if Nzt == 1
        dzt = 0;
        set(findobj(gcf,'Tag','dzt'),'String',num2str(dzt));
        warndlg('For a single element, array spacing is set zero.',...
                 '  Array Spacing','help');
     elseif Nzt > 1
       dzt = str2num(get(findobj(gcf,'Tag','dzt'),'String'));
        if dzt == 0
           set(findobj(gcf,'Tag','dzt'),'String',num2str(1));
           warndlg('For the number of elements, array spacing cannot be
zero.',...
              '  Array Spacing','help');
        end
     end

  case 'dxt'
     h_Dxt = findobj(gcf,'Tag','dxt');
     dxt_str = get(h_Dxt,'String');
     dxt = getSpacing(dxt_str);
     set(h_Dxt,'String',num2str(dxt));
     if dxt == 0
        Nxt = 1;
        set(findobj(gcf,'Tag','Nxt'),'String',num2str(Nxt));
        warndlg('Number of array elements set to 1.',' Number of Array
Elements','help');
     end

     Nxt = str2num(get(findobj(gcf,'Tag','Nxt'),'String'));
     if dxt > 0
        if Nxt == 1
           errordlg('For the spacing indicated, number of elements must be at
least 2.',...
                  'Number of Elements','error');
           dxt = 0;
           set(h_Dxt,'String',num2str(dxt));
        end
     end

  case 'dzt'
     h_Dzt = findobj(gcf,'Tag','dzt');
     dzt_str = get(h_Dzt,'String');
     dzt = getSpacing(dzt_str);
     set(h_Dzt,'String',num2str(dzt));
     if dzt == 0
```

```
         Nzt = 1;
         set(findobj(gcf,'Tag','Nzt'),'String',num2str(Nzt));
         warndlg('Number of array elements set to 1.',...
                  ' Number of Array Elements','warn');
      end

      Nzt = str2num(get(findobj(gcf,'Tag','Nzt'),'String'));
      if dzt > 0
         if Nzt == 1
            errordlg('For the spacing indicated, number of elements must be at
least 2.',...
                     'Number of Elements','error');
            dzt = 0;
            set(h_Dzt,'String',num2str(dzt));
         end
      end

   case 'Mxt'
      h_Mxt = findobj(gcf,'Tag','Mxt');
      mxt_str = get(h_Mxt,'String');
      Mxt = getNEL(mxt_str);
      set(h_Mxt,'String',num2str(Mxt));
      if Mxt == 1
         lxt = 0;
         set(findobj(gcf,'Tag','lxt'),'String',num2str(lxt));
         warndlg('For a single element, array spacing is set to zero.',...
                  '  Array Spacing','help');
      elseif Mxt > 1
        lxt = str2num(get(findobj(gcf,'Tag','lxt'),'String'));
         if lxt == 0
            set(findobj(gcf,'Tag','lxt'),'String',num2str(1));
            warndlg('For the number of elements, array spacing cannot be
zero.',...
               '  Array Spacing','help');
         end
      end

   case 'Mzt'
      h_Mzt = findobj(gcf,'Tag','Mzt');
      mzt_str = get(h_Mzt,'String');
      Mzt = getNEL(mzt_str);
      set(h_Mzt,'String',num2str(Mzt));
      if Mzt == 1
         lzt = 0;
         set(findobj(gcf,'Tag','lzt'),'String',num2str(lzt));
         warndlg('For a single element, array spacing is set zero.',...
                  '  Array Spacing','help');
      elseif Mzt > 1
        lzt = str2num(get(findobj(gcf,'Tag','lzt'),'String'));
         if lzt == 0
            set(findobj(gcf,'Tag','lzt'),'String',num2str(1));
            warndlg('For the number of elements, array spacing cannot be
zero.',...
               '  Array Spacing','help');
         end
      end

   case 'lxt'
      h_Lxt = findobj(gcf,'Tag','lxt');
      lxt_str = get(h_Lxt,'String');
      lxt = getSpacing(lxt_str);
      set(h_Lxt,'String',num2str(lxt));
```

```
      if lxt == 0
         Mxt = 1;
         set(findobj(gcf,'Tag','Mxt'),'String',num2str(Mxt));
         warndlg('Number of array elements set to 1.',' Number of Array 
Elements','help');
      end

      Mxt = str2num(get(findobj(gcf,'Tag','Mxt'),'String'));
      if lxt > 0
         if Mxt == 1
            errordlg('For the spacing indicated, number of elements must be at 
least 2.',...
                       'Number of Elements','error');
            lxt = 0;
            set(h_Lxt,'String',num2str(lxt));
         end
      end

   case 'lzt'
      h_Lzt = findobj(gcf,'Tag','lzt');
      lzt_str = get(h_Lzt,'String');
      lzt = getSpacing(lzt_str);
      set(h_Lzt,'String',num2str(lzt));
      if lzt == 0
         Mzt = 1;
         set(findobj(gcf,'Tag','Mzt'),'String',num2str(Mzt));
         warndlg('Number of array elements set to 1.',...
                  ' Number of Array Elements','warn');
      end

      Mzt = str2num(get(findobj(gcf,'Tag','Mzt'),'String'));
      if lzt > 0
         if Mzt == 1
            errordlg('For the spacing indicated, number of elements must be at 
least 2.',...
                       'Number of Elements','error');
            lzt = 0;
            set(h_Lzt,'String',num2str(lzt));
         end
      end

   case 'txs_xdist'
      h_txs_xdist = findobj(gcf,'Tag','txs_xdist');
      txs_xdist_val = get(h_txs_xdist,'Value');
      ixdist = txs_xdist_val;
      if (ixdist == 2 | ixdist == 4)
        Mxt = str2num(get(findobj(gcf,'Tag','Mxt'),'String'));
         if rem(Mxt,2) ~= 0
            set(findobj(gcf,'Tag','txs_xdist'),'Value',1);
            errordlg('Number of subarrays in X-plane must be even. Change 
number of subarrays or select another distribution.', ...
                  ' Number of Subarrays','error');
         end
      end

      if (ixdist == 2) |(ixdist == 3) |(ixdist == 4)
         set(findobj(gcf,'Tag','txs_peddbx'),'Enable','on');
         set(findobj(gcf,'Tag','txs_nexpx'),'Enable','on');
      else
         set(findobj(gcf,'Tag','txs_peddbx'),'Enable','off');
         set(findobj(gcf,'Tag','txs_nexpx'),'Enable','off');
      end
```

```
   case 'txs_zdist'
      h_txs_zdist = findobj(gcf,'Tag','txs_zdist');
      txs_zdist_val = get(h_txs_zdist,'Value');
      izdist = txs_zdist_val;
      if (izdist == 2 | izdist == 4)
        Mzt = str2num(get(findobj(gcf,'Tag','Mzt'),'String'));
         if rem(Mzt,2) ~= 0
            set(findobj(gcf,'Tag','txs_zdist'),'Value',1);
            errordlg('Number of subarrays in Z-plane must be even. Change
number of subarrays or select another distribution.', ...
               ' Number of Subarrays','error');
         end
      end

      if (izdist == 2) |(izdist == 3) |(izdist == 4)
         set(findobj(gcf,'Tag','txs_peddbz'),'Enable','on');
         set(findobj(gcf,'Tag','txs_nexpz'),'Enable','on');
      else
         set(findobj(gcf,'Tag','txs_peddbz'),'Enable','off');
         set(findobj(gcf,'Tag','txs_nexpz'),'Enable','off');
      end

   case 'tx_xdist'
      h_tx_xdist = findobj(gcf,'Tag','tx_xdist');
      tx_xdist_val = get(h_tx_xdist,'Value');
      ixdist = tx_xdist_val;
      if (ixdist == 2 | ixdist == 4)
        Nxt = str2num(get(findobj(gcf,'Tag','Nxt'),'String'));
         if rem(Nxt,2) ~= 0
            set(findobj(gcf,'Tag','tx_xdist'),'Value',1);
            errordlg('Number of elements in X-plane must be even. Change number
of elements or select another distribution.', ...
               ' Number of Subrray Elements','error');
         end
      end

      if (ixdist == 2) |(ixdist == 3) |(ixdist == 4)
         set(findobj(gcf,'Tag','tx_peddbx'),'Enable','on');
         set(findobj(gcf,'Tag','tx_nexpx'),'Enable','on');
      else
         set(findobj(gcf,'Tag','tx_peddbx'),'Enable','off');
         set(findobj(gcf,'Tag','tx_nexpx'),'Enable','off');
      end

    case 'tx_zdist'
      h_tx_zdist = findobj(gcf,'Tag','tx_zdist');
      tx_zdist_val = get(h_tx_zdist,'Value');
      izdist = tx_zdist_val;
      if (izdist == 2 | izdist == 4)
        Nzt = str2num(get(findobj(gcf,'Tag','Nzt'),'String'));
         if rem(Nzt,2) ~= 0
            set(findobj(gcf,'Tag','tx_zdist'),'Value',1);
            errordlg('Number of elements in Z-plane must be even. Change number
of elements or select another distribution.', ...
               ' Number of Subrray Elements','error');
         end
      end

      if (izdist == 2) |(izdist == 3) |(izdist == 4)
         set(findobj(gcf,'Tag','tx_peddbz'),'Enable','on');
         set(findobj(gcf,'Tag','tx_nexpz'),'Enable','on');
```

```
      else
         set(findobj(gcf,'Tag','tx_peddbz'),'Enable','off');
         set(findobj(gcf,'Tag','tx_nexpz'),'Enable','off');
      end

   %Case for Receive Array parameters
   case 'Nxr'
      h_Nxr = findobj(gcf,'Tag','Nxr');
      nxr_str = get(h_Nxr,'String');
      Nxr = getNEL(nxr_str);
      set(h_Nxr,'String',num2str(Nxr));
      if Nxr == 1
         dxr = 0;
         set(findobj(gcf,'Tag','dxr'),'String',num2str(dxr));
         warndlg('For a single element, array spacing is set to zero.',...
                  '  Array Spacing','help');
      elseif Nxr > 1
        dxr = str2num(get(findobj(gcf,'Tag','dxr'),'String'));
         if dxr == 0
            set(findobj(gcf,'Tag','dxr'),'String',num2str(1));
            warndlg('For the number of elements, array spacing cannot be
zero.',...
               '  Array Spacing','help');
         end
      end

   case 'Nzr'
      h_Nzr = findobj(gcf,'Tag','Nzr');
      nzr_str = get(h_Nzr,'String');
      Nzr = getNEL(nzr_str);
      set(h_Nzr,'String',num2str(Nzr));
      if Nzr == 1
         dzr = 0;
         set(findobj(gcf,'Tag','dzr'),'String',num2str(dzr));
         warndlg('For a single element, array spacing is set zero.',...
                  '  Array Spacing','help');
      elseif Nzr > 1
        dzr = str2num(get(findobj(gcf,'Tag','dzr'),'String'));
         if dzr == 0
            set(findobj(gcf,'Tag','dzr'),'String',num2str(1));
            warndlg('For the number of elements, array spacing cannot be
zero.',...
               '  Array Spacing','help');
         end
      end

   case 'dxr'
      h_Dxr = findobj(gcf,'Tag','dxr');
      dxr_str = get(h_Dxr,'String');
      dxr = getSpacing(dxr_str);
      set(h_Dxr,'String',num2str(dxr));
      if dxr == 0
         Nxr = 1;
         set(findobj(gcf,'Tag','Nxr'),'String',num2str(Nxr));
         warndlg('Number of array elements set to 1.',' Number of Array
Elements','help');
      end

      Nxr = str2num(get(findobj(gcf,'Tag','Nxr'),'String'));
      if dxr > 0
         if Nxr == 1
```

```
            errordlg('For the spacing indicated, number of elements must be at
least 2.',...
                   'Number of Elements','error');
            dxr = 0;
            set(h_Dxr,'String',num2str(dxr));
         end
      end

   case 'dzr'
      h_Dzr = findobj(gcf,'Tag','dzr');
      dzr_str = get(h_Dzr,'String');
      dzr = getSpacing(dzr_str);
      set(h_Dzr,'String',num2str(dzr));
      if dzr == 0
         Nzr = 1;
         set(findobj(gcf,'Tag','Nzr'),'String',num2str(Nzr));
         warndlg('Number of array elements set to 1.',...
                  ' Number of Array Elements','warn');
      end

      Nzr = str2num(get(findobj(gcf,'Tag','Nzr'),'String'));
      if dzr > 0
         if Nzr == 1
            errordlg('For the spacing indicated, number of elements must be at
least 2.',...
                   'Number of Elements','error');
            dzr = 0;
            set(h_Dzr,'String',num2str(dzr));
         end
      end

   case 'Mxr'
      h_Mxr = findobj(gcf,'Tag','Mxr');
      mxr_str = get(h_Mxr,'String');
      Mxr = getNEL(mxr_str);
      set(h_Mxr,'String',num2str(Mxr));
      if Mxr == 1
         lxr = 0;
         set(findobj(gcf,'Tag','lxr'),'String',num2str(lxr));
         warndlg('For a single element, array spacing is set to zero.',...
                  '  Array Spacing','help');
      elseif Mxr > 1
        lxr = str2num(get(findobj(gcf,'Tag','lxr'),'String'));
         if lxr == 0
            set(findobj(gcf,'Tag','lxr'),'String',num2str(1));
            warndlg('For the number of elements, array spacing cannot be
zero.',...
               '  Array Spacing','help');
         end
      end

   case 'Mzr'
      h_Mzr = findobj(gcf,'Tag','Mzr');
      mzr_str = get(h_Mzr,'String');
      Mzr = getNEL(mzr_str);
      set(h_Mzr,'String',num2str(Mzr));
      if Mzr == 1
         lzr = 0;
         set(findobj(gcf,'Tag','lzr'),'String',num2str(lzr));
         warndlg('For a single element, array spacing is set zero.',...
                  '  Array Spacing','help');
      elseif Mzr > 1
```

```
        lzr = str2num(get(findobj(gcf,'Tag','lzr'),'String'));
         if lzr == 0
            set(findobj(gcf,'Tag','lzr'),'String',num2str(1));
            warndlg('For the number of elements, array spacing cannot be
zero.',...
               '  Array Spacing','help');
         end
      end

   case 'lxr'
      h_Lxr = findobj(gcf,'Tag','lxr');
      lxr_str = get(h_Lxr,'String');
      lxr = getSpacing(lxr_str);
      set(h_Lxr,'String',num2str(lxr));
      if lxr == 0
         Mxr = 1;
         set(findobj(gcf,'Tag','Mxr'),'String',num2str(Mxr));
         warndlg('Number of array elements set to 1.',' Number of Array
Elements','help');
      end

      Mxr = str2num(get(findobj(gcf,'Tag','Mxr'),'String'));
      if lxr > 0
         if Mxr == 1
            errordlg('For the spacing indicated, number of elements must be at
least 2.',...
                   'Number of Elements','error');
            lxr = 0;
            set(h_Lxr,'String',num2str(lxr));
         end
      end

   case 'lzr'
      h_Lzr = findobj(gcf,'Tag','lzr');
      lzr_str = get(h_Lzr,'String');
      lzr = getSpacing(lzr_str);
      set(h_Lzr,'String',num2str(lzr));
      if lzr == 0
         Mzr = 1;
         set(findobj(gcf,'Tag','Mzr'),'String',num2str(Mzr));
         warndlg('Number of array elements set to 1.',...
                  ' Number of Array Elements','warn');
      end

      Mzr = str2num(get(findobj(gcf,'Tag','Mzr'),'String'));
      if lzr > 0
         if Mzr == 1
            errordlg('For the spacing indicated, number of elements must be at
least 2.',...
                   'Number of Elements','error');
            lzr = 0;
            set(h_Lzr,'String',num2str(lzr));
         end
      end

    case 'rxs_xdist'
      h_rxs_xdist = findobj(gcf,'Tag','rxs_xdist');
      rxs_xdist_val = get(h_rxs_xdist,'Value');
      ixdist = rxs_xdist_val;
      if (ixdist == 2 | ixdist == 4)
        Mxr = str2num(get(findobj(gcf,'Tag','Mxr'),'String'));
         if rem(Mxr,2) ~= 0
```

```
            set(findobj(gcf,'Tag','rxs_xdist'),'Value',1);
            errordlg('Number of subarrays in X-plane must be even. Change
number of subarrays or select another distribution.', ...
               ' Number of Subarrays','error');
          end
      end

      if (ixdist == 2) |(ixdist == 3) |(ixdist == 4)
         set(findobj(gcf,'Tag','rxs_peddbx'),'Enable','on');
         set(findobj(gcf,'Tag','rxs_nexpx'),'Enable','on');
      else
         set(findobj(gcf,'Tag','rxs_peddbx'),'Enable','off');
         set(findobj(gcf,'Tag','rxs_nexpx'),'Enable','off');
      end

    case 'rxs_zdist'
        h_rxs_zdist = findobj(gcf,'Tag','rxs_zdist');
        rxs_zdist_val = get(h_rxs_zdist,'Value');
        izdist = rxs_zdist_val;
        if (izdist == 2 | izdist == 4)
          Mzr = str2num(get(findobj(gcf,'Tag','Mzr'),'String'));
          if rem(Mzr,2) ~= 0
            set(findobj(gcf,'Tag','rxs_zdist'),'Value',1);
            errordlg('Number of subarrays in Z-plane must be even. Change
number of subarrays or select another distribution.', ...
               ' Number of Subarrays','error');
          end
        end

        if (izdist == 2) |(izdist == 3) |(izdist == 4)
          set(findobj(gcf,'Tag','rxs_peddbz'),'Enable','on');
          set(findobj(gcf,'Tag','rxs_nexpz'),'Enable','on');
        else
          set(findobj(gcf,'Tag','rxs_peddbz'),'Enable','off');
          set(findobj(gcf,'Tag','rxs_nexpz'),'Enable','off');
        end

    case 'rx_xdist'
        h_rx_xdist = findobj(gcf,'Tag','rx_xdist');
        rx_xdist_val = get(h_rx_xdist,'Value');
        ixdist = rx_xdist_val;
        if (ixdist == 2 | ixdist == 4)
           Nxr = str2num(get(findobj(gcf,'Tag','Nxr'),'String'));
           if rem(Nxr,2) ~= 0
            set(findobj(gcf,'Tag','rx_xdist'),'Value',1);
            errordlg('Number of elements in X-plane must be even. Change number
of elements or select another distribution.', ...
               ' Number of Subrray Elements','error');
           end
        end

        if (ixdist == 2) |(ixdist == 3) |(ixdist == 4)
           set(findobj(gcf,'Tag','rx_peddbx'),'Enable','on');
           set(findobj(gcf,'Tag','rx_nexpx'),'Enable','on');
        else
           set(findobj(gcf,'Tag','rx_peddbx'),'Enable','off');
           set(findobj(gcf,'Tag','rx_nexpx'),'Enable','off');
        end

    case 'rx_zdist'
        h_rx_zdist = findobj(gcf,'Tag','rx_zdist');
        rx_zdist_val = get(h_rx_zdist,'Value');
```

```
        izdist = rx_zdist_val;
        if (izdist == 2 | izdist == 4)
           Nzr = str2num(get(findobj(gcf,'Tag','Nzr'),'String'));
           if rem(Nzr,2) ~= 0
            set(findobj(gcf,'Tag','rx_zdist'),'Value',1);
            errordlg('Number of elements in Z-plane must be even. Change number
of elements or select another distribution.', ...
               ' Number of Subrray Elements','error');
           end
        end

        if (izdist == 2) |(izdist == 3) |(izdist == 4)
           set(findobj(gcf,'Tag','rx_peddbz'),'Enable','on');
           set(findobj(gcf,'Tag','rx_nexpz'),'Enable','on');
        else
           set(findobj(gcf,'Tag','rx_peddbz'),'Enable','off');
           set(findobj(gcf,'Tag','rx_nexpz'),'Enable','off');
        end

    %Case Calculate and plot two-way pattern
    case 'Calculate'

        rad=pi/180;

        %Get parameters
        h_eltype = get(findobj(gcf,'Tag','eltype'),'Value');
            if h_eltype == 1, diptype = 'h'; end
            if h_eltype == 2, diptype = 's'; end
            if h_eltype == 3, diptype = 'i'; end
        h_dipdir = get(findobj(gcf,'Tag','dipdir'),'Value');
            if h_dipdir == 1, dipdir = 'x'; end
            if h_dipdir == 2, dipdir = 'z'; end
        f = (str2num(get(findobj(gcf,'Tag','frequency'),'String')))*1e6;
        h = str2num(get(findobj(gcf,'Tag','height'),'String'));
        thetas = str2num(get(findobj(gcf,'Tag','thetas'),'String'));
        phis = str2num(get(findobj(gcf,'Tag','phis'),'String'));
        tstart = str2num(get(findobj(gcf,'Tag','tstart'),'String'));
        tstop = str2num(get(findobj(gcf,'Tag','tstop'),'String'));
        pstart = str2num(get(findobj(gcf,'Tag','pstart'),'String'));
        pstop = str2num(get(findobj(gcf,'Tag','pstop'),'String'));
        delt = str2num(get(findobj(gcf,'Tag','delt'),'String'));
        delp = str2num(get(findobj(gcf,'Tag','delp'),'String'));

        Nxt = str2num(get(findobj(gcf,'Tag','Nxt'),'String'));
        Nzt = str2num(get(findobj(gcf,'Tag','Nzt'),'String'));
        dxt = str2num(get(findobj(gcf,'Tag','dxt'),'String'));
        dzt = str2num(get(findobj(gcf,'Tag','dzt'),'String'));
        Mxt = str2num(get(findobj(gcf,'Tag','Mxt'),'String'));
        Mzt = str2num(get(findobj(gcf,'Tag','Mzt'),'String'));
        lxt = str2num(get(findobj(gcf,'Tag','lxt'),'String'));
        lzt = str2num(get(findobj(gcf,'Tag','lzt'),'String'));

        Nxr = str2num(get(findobj(gcf,'Tag','Nxr'),'String'));
        Nzr = str2num(get(findobj(gcf,'Tag','Nzr'),'String'));
        dxr = str2num(get(findobj(gcf,'Tag','dxr'),'String'));
        dzr = str2num(get(findobj(gcf,'Tag','dzr'),'String'));
        Mxr = str2num(get(findobj(gcf,'Tag','Mxr'),'String'));
        Mzr = str2num(get(findobj(gcf,'Tag','Mzr'),'String'));
        lxr = str2num(get(findobj(gcf,'Tag','lxr'),'String'));
        lzr = str2num(get(findobj(gcf,'Tag','lzr'),'String'));

        txs_xdist = get(findobj(gcf,'Tag','txs_xdist'),'Value');
```

```
txs_peddbxval = get(findobj(gcf,'Tag','txs_peddbx'),'Value');
    if txs_peddbxval==1, txs_peddbx=15; end
    if txs_peddbxval==2, txs_peddbx=20; end
    if txs_peddbxval==3, txs_peddbx=25; end
    if txs_peddbxval==4, txs_peddbx=30; end
    if txs_peddbxval==5, txs_peddbx=35; end
    if txs_peddbxval==6, txs_peddbx=40; end
    if txs_peddbxval==7, txs_peddbx=45; end
    if txs_peddbxval==8, txs_peddbx=50; end
txs_nexpx = get(findobj(gcf,'Tag','txs_nexpx'),'Value');

txs_zdist = get(findobj(gcf,'Tag','txs_zdist'),'Value');
txs_peddbzval = get(findobj(gcf,'Tag','txs_peddbz'),'Value');
    if txs_peddbzval==1, txs_peddbz=15; end
    if txs_peddbzval==2, txs_peddbz=20; end
    if txs_peddbzval==3, txs_peddbz=25; end
    if txs_peddbzval==4, txs_peddbz=30; end
    if txs_peddbzval==5, txs_peddbz=35; end
    if txs_peddbzval==6, txs_peddbz=40; end
    if txs_peddbzval==7, txs_peddbz=45; end
    if txs_peddbzval==8, txs_peddbz=50; end
txs_nexpz = get(findobj(gcf,'Tag','txs_nexpz'),'Value');

tx_xdist = get(findobj(gcf,'Tag','tx_xdist'),'Value');
tx_peddbxval = get(findobj(gcf,'Tag','tx_peddbx'),'Value');
    if tx_peddbxval==1, tx_peddbx=15; end
    if tx_peddbxval==2, tx_peddbx=20; end
    if tx_peddbxval==3, tx_peddbx=25; end
    if tx_peddbxval==4, tx_peddbx=30; end
    if tx_peddbxval==5, tx_peddbx=35; end
    if tx_peddbxval==6, tx_peddbx=40; end
    if tx_peddbxval==7, tx_peddbx=45; end
    if tx_peddbxval==8, tx_peddbx=50; end
tx_nexpx = get(findobj(gcf,'Tag','tx_nexpx'),'Value');

tx_zdist = get(findobj(gcf,'Tag','tx_zdist'),'Value');
tx_peddbzval = get(findobj(gcf,'Tag','tx_peddbz'),'Value');
    if tx_peddbzval==1, tx_peddbz=15; end
    if tx_peddbzval==2, tx_peddbz=20; end
    if tx_peddbzval==3, tx_peddbz=25; end
    if tx_peddbzval==4, tx_peddbz=30; end
    if tx_peddbzval==5, tx_peddbz=35; end
    if tx_peddbzval==6, tx_peddbz=40; end
    if tx_peddbzval==7, tx_peddbz=45; end
    if tx_peddbzval==8, tx_peddbz=50; end
tx_nexpz = get(findobj(gcf,'Tag','tx_nexpz'),'Value');

txs_ampx = getamplitudes(Mxt,txs_xdist,txs_peddbx,txs_nexpx);
txs_ampz = getamplitudes(Mzt,txs_zdist,txs_peddbz,txs_nexpz);
tx_ampx = getamplitudes(Nxt,tx_xdist,tx_peddbx,tx_nexpx);
tx_ampz = getamplitudes(Nzt,tx_zdist,tx_peddbz,tx_nexpz);

rxs_xdist = get(findobj(gcf,'Tag','rxs_xdist'),'Value');
rxs_peddbxval = get(findobj(gcf,'Tag','rxs_peddbx'),'Value');
    if rxs_peddbxval==1, rxs_peddbx=15; end
    if rxs_peddbxval==2, rxs_peddbx=20; end
    if rxs_peddbxval==3, rxs_peddbx=25; end
    if rxs_peddbxval==4, rxs_peddbx=30; end
    if rxs_peddbxval==5, rxs_peddbx=35; end
    if rxs_peddbxval==6, rxs_peddbx=40; end
    if rxs_peddbxval==7, rxs_peddbx=45; end
    if rxs_peddbxval==8, rxs_peddbx=50; end
```

```
        rxs_nexpx = get(findobj(gcf,'Tag','rxs_nexpx'),'Value');

        rxs_zdist = get(findobj(gcf,'Tag','rxs_zdist'),'Value');
        rxs_peddbzval = get(findobj(gcf,'Tag','rxs_peddbz'),'Value');
            if rxs_peddbzval==1, rxs_peddbz=15; end
            if rxs_peddbzval==2, rxs_peddbz=20; end
            if rxs_peddbzval==3, rxs_peddbz=25; end
            if rxs_peddbzval==4, rxs_peddbz=30; end
            if rxs_peddbzval==5, rxs_peddbz=35; end
            if rxs_peddbzval==6, rxs_peddbz=40; end
            if rxs_peddbzval==7, rxs_peddbz=45; end
            if rxs_peddbzval==8, rxs_peddbz=50; end
        rxs_nexpz = get(findobj(gcf,'Tag','rxs_nexpz'),'Value');

        rx_xdist = get(findobj(gcf,'Tag','rx_xdist'),'Value');
        rx_peddbxval = get(findobj(gcf,'Tag','rx_peddbx'),'Value');
            if rx_peddbxval==1, rx_peddbx=15; end
            if rx_peddbxval==2, rx_peddbx=20; end
            if rx_peddbxval==3, rx_peddbx=25; end
            if rx_peddbxval==4, rx_peddbx=30; end
            if rx_peddbxval==5, rx_peddbx=35; end
            if rx_peddbxval==6, rx_peddbx=40; end
            if rx_peddbxval==7, rx_peddbx=45; end
            if rx_peddbxval==8, rx_peddbx=50; end
        rx_nexpx = get(findobj(gcf,'Tag','rx_nexpx'),'Value');

        rx_zdist = get(findobj(gcf,'Tag','rx_zdist'),'Value');
        rx_peddbzval = get(findobj(gcf,'Tag','rx_peddbz'),'Value');
            if rx_peddbzval==1, rx_peddbz=15; end
            if rx_peddbzval==2, rx_peddbz=20; end
            if rx_peddbzval==3, rx_peddbz=25; end
            if rx_peddbzval==4, rx_peddbz=30; end
            if rx_peddbzval==5, rx_peddbz=35; end
            if rx_peddbzval==6, rx_peddbz=40; end
            if rx_peddbzval==7, rx_peddbz=45; end
            if rx_peddbzval==8, rx_peddbz=50; end
        rx_nexpz = get(findobj(gcf,'Tag','rx_nexpz'),'Value');

        rxs_ampx = getamplitudes(Mxr,rxs_xdist,rxs_peddbx,rxs_nexpx);
        rxs_ampz = getamplitudes(Mzr,rxs_zdist,rxs_peddbz,rxs_nexpz);
        rx_ampx = getamplitudes(Nxr,rx_xdist,rx_peddbx,rx_nexpx);
        rx_ampz = getamplitudes(Nzr,rx_zdist,rx_peddbz,rx_nexpz);

        % Compute CAF of Transmit DSA
        if diptype == 'h' %compute for half wave dipoles

[TxSAFtheta,TxSAFphi,TxDAFtheta,TxDAFphi,Ut,Wt]=caf_hdip2('t',f,dxt,Nxt,dzt,Nzt
,lxt,Mxt,lzt,Mzt,h,dipdir,thetas,phis,tstart,tstop,pstart,pstop,delt,delp,txs_a
mpx,txs_ampz,tx_ampx,tx_ampz);
        end

        if diptype == 's' %compute for short dipoles

[TxSAFtheta,TxSAFphi,TxDAFtheta,TxDAFphi,Ut,Wt]=caf_sdip2('t',f,dxt,Nxt,dzt,Nzt
,lxt,Mxt,lzt,Mzt,h,dipdir,thetas,phis,tstart,tstop,pstart,pstop,delt,delp,txs_a
mpx,txs_ampz,tx_ampx,tx_ampz);
        end

        if diptype == 'i' %compute for isotropic elements

[TxSAFtheta,TxSAFphi,TxDAFtheta,TxDAFphi,Ut,Wt]=caf_iso2('t',f,dxt,Nxt,dzt,Nzt,
```

```
lxt,Mxt,lzt,Mzt,h,thetas,phis,tstart,tstop,pstart,pstop,delt,delp,txs_ampx,txs_
ampz,tx_ampx,tx_ampz);
        end

        % Normalize transmit DSA pattern
        MAXtx=max([max(max(abs(TxDAFtheta))),max(max(abs(TxDAFphi)))]);
        TxDAFt_norm=abs(TxDAFtheta)/MAXtx; %Find magnitude and normalize
        TxDAFp_norm=abs(TxDAFphi)/MAXtx;
        TxDAFt_db=20*log10(TxDAFt_norm); %Compute in dB
        TxDAFp_db=20*log10(TxDAFp_norm);

        % Compute CAF of Receive DSA
        if diptype == 'h' %Compute for half wave dipoles
            
[RxSAFtheta,RxSAFphi,RxDAFtheta,RxDAFphi,Ur,Wr]=caf_hdip2('r',f,dxr,Nxr,dzr,Nzr
,lxr,Mxr,lzr,Mzr,h,dipdir,thetas,phis,tstart,tstop,pstart,pstop,delt,delp,rxs_a
mpx,rxs_ampz,rx_ampx,rx_ampz);
        end

        if diptype == 's' %Compute for short dipoles
            
[RxSAFtheta,RxSAFphi,RxDAFtheta,RxDAFphi,Ur,Wr]=caf_sdip2('r',f,dxr,Nxr,dzr,Nzr
,lxr,Mxr,lzr,Mzr,h,dipdir,thetas,phis,tstart,tstop,pstart,pstop,delt,delp,rxs_a
mpx,rxs_ampz,rx_ampx,rx_ampz);
        end

        if diptype == 'i' %Compute for isotropic elements
            
[RxSAFtheta,RxSAFphi,RxDAFtheta,RxDAFphi,Ur,Wr]=caf_iso2('r',f,dxr,Nxr,dzr,Nzr,
lxr,Mxr,lzr,Mzr,h,thetas,phis,tstart,tstop,pstart,pstop,delt,delp,rxs_ampx,rxs_
ampz,rx_ampx,rx_ampz);
        end

        % Normalize receive DSA pattern
        MAXrx=max([max(max(abs(RxDAFtheta))),max(max(abs(RxDAFphi)))]);
        RxDAFt_norm=abs(RxDAFtheta)/MAXrx; %Find magnitude and normalize
        RxDAFp_norm=abs(RxDAFphi)/MAXrx;
        RxDAFt_db=20*log10(RxDAFt_norm); %Compute in dB
        RxDAFp_db=20*log10(RxDAFp_norm);

        % Compute Two-way Pattern of Transmit and Receive DSAs
        Two_way_DAFt=TxDAFtheta.*RxDAFtheta;
        Two_way_DAFp=TxDAFphi.*RxDAFphi;

        MAXtw=max([max(max(abs(Two_way_DAFt))),max(max(abs(Two_way_DAFp)))]);
        Two_way_DAFt_norm=abs(Two_way_DAFt)/MAXtw; %Normalized theta pattern
        Two_way_DAFp_norm=abs(Two_way_DAFp)/MAXtw; %Normalized phi pattern

        Two_way_DAFt_db=20*log10(Two_way_DAFt_norm); %Compute in dB
        Two_way_DAFp_db=20*log10(Two_way_DAFp_norm);

        %Export dsa configuration to file dsaconfig.m
        save dsaconfig f diptype dipdir h delt delp thetas phis ...
            tstart tstop pstart pstop ...
            Nxt Nzt dxt dzt Mxt Mzt lxt lzt ...
            Nxr Nzr dxr dzr Mxr Mzr lxr lzr ...
            txs_xdist txs_peddbx txs_nexpx ...
            txs_zdist txs_peddbz txs_nexpz ...
            tx_xdist tx_peddbx tx_nexpx ...
            tx_zdist tx_peddbz tx_nexpz ...
            rxs_xdist rxs_peddbx rxs_nexpx ...
            rxs_zdist rxs_peddbz rxs_nexpz ...
```

```
        rx_xdist rx_peddbx rx_nexpx ...
        rx_zdist rx_peddbz rx_nexpz ...

    %Export dsa pattern data to file dsapattern.m
    save dsapattern Ur Wr TxDAFt_db TxDAFp_db RxDAFt_db RxDAFp_db...
        Two_way_DAFt_db Two_way_DAFp_db

    %Set plot dynamic range to pmin
    pmin = -80; %Set the pmin value (dB)
    ip = floor((pstop-pstart)/delp)+1;
    it = floor((tstop-tstart)/delt)+1;
    for i1 = 1:ip
        for i2 = 1:it
            theta(i1,i2) = tstart + (i2-1)*delt;
            phi(i1,i2) = pstart + (i1-1)*delp;
        end
    end

    if ip == 1 %Plot phi cut
        figure(1),clf
        subplot(211)
        plot(theta,TxDAFt_db),grid %Plot transmit DSA theta component
        axis([tstart,tstop,pmin,0])
        title(['Transmit DSA Pattern, \phi=',num2str(pstart),'^o cut'])
        xlabel('Pattern Angle, \theta (deg)')
        ylabel('Normalized |F_\theta| (dB)')
        subplot(212)
        plot(theta,TxDAFp_db),grid %Plot transmit DSA phi component
        axis([tstart,tstop,pmin,0])
        title(['Transmit DSA Pattern, \phi=',num2str(pstart),'^o cut'])
        xlabel('Pattern Angle, \theta (deg)')
        ylabel('Normalized |F_\phi| (dB)')


        figure(2),clf
        subplot(211)
        plot(theta,RxDAFt_db),grid %Plot receive DSA theta component
        axis([tstart,tstop,pmin,0])
        title(['Receive DSA Pattern, \phi=',num2str(pstart),'^o cut'])
        xlabel('Pattern Angle, \theta (deg)')
        ylabel('Normalized |F_\theta| (dB)')
        subplot(212)
        plot(theta,RxDAFp_db),grid %Plot receive DSA phi component
        axis([tstart,tstop,pmin,0])
        title(['Receive DSA Pattern, \phi=',num2str(pstart),'^o cut'])
        xlabel('Pattern Angle, \theta (deg)')
        ylabel('Normalized |F_\phi| (dB)')

        figure(3),clf
        subplot(211)
        plot(theta,Two_way_DAFt_db),grid %Plot two-way theta component
        axis([tstart,tstop,pmin,0])
        title(['Two-way Pattern, \phi=',num2str(pstart),'^o cut'])
        xlabel('Pattern Angle, \theta (deg)')
        ylabel('Normalized |F_\theta| (dB)')
        subplot(212)
        plot(theta,Two_way_DAFp_db),grid %Plot two-way phi component
        axis([tstart,tstop,pmin,0])
        title(['Two-way Pattern |F_n_o_r_m(\phi)|,
\phi=',num2str(pstart),'^o cut'])
        xlabel('Pattern Angle, \theta (deg)')
        ylabel('Normalized |F_\phi| (dB)')
```

```
end

if it == 1 %Plot theta cut
    figure(1),clf
    subplot(211)
    plot(phi,TxDAFt_db),grid %Plot transmit DSA theta component
    axis([pstart,pstop,pmin,0])
    title(['Transmit DSA Pattern, \theta=',num2str(tstart),'^o cut'])
    xlabel('Pattern Angle, \phi (deg)')
    ylabel('Normalized |F_\theta| (dB)')
    subplot(212)
    plot(phi,TxDAFp_db),grid %Plot transmit DSA phi component
    axis([pstart,pstop,pmin,0])
    title(['Transmit DSA Pattern, \theta=',num2str(tstart),'^o cut'])
    xlabel('Pattern Angle, \phi (deg)')
    ylabel('Normalized |F_\phi| (dB)')

    figure(2),clf
    subplot(211)
    plot(phi,RxDAFt_db),grid %Plot receive DSA theta component
    axis([pstart,pstop,pmin,0])
    title(['Receive DSA Pattern, \theta=',num2str(tstart),'^o cut'])
    xlabel('Pattern Angle, \phi (deg)')
    ylabel('Normalized |F_\theta| (dB)')
    subplot(212)
    plot(phi,RxDAFp_db),grid %Plot receive DSA phi component
    axis([pstart,pstop,pmin,0])
    title(['Receive DSA Pattern, \theta=',num2str(tstart),'^o cut'])
    xlabel('Pattern Angle, \phi (deg)')
    ylabel('Normalized |F_\phi| (dB)')

    figure(3),clf
    subplot(211)
    plot(phi,Two_way_DAFt_db),grid %Plot two-way theta component
    axis([pstart,pstop,pmin,0])
    title(['Two-way Pattern, \theta=',num2str(tstart),'^o cut'])
    xlabel('Pattern Angle, \phi (deg)')
    ylabel('Normalized |F_\theta| (dB)')
    subplot(212)
    plot(phi,Two_way_DAFp_db),grid %Plot two-way phi component
    axis([pstart,pstop,pmin,0])
    title(['Two-way Pattern, \theta=',num2str(tstart),'^o cut'])
    xlabel('Pattern Angle, \phi (deg)')
    ylabel('Normalized |F_\phi| (dB)')
end

if ip>1 & it>1

    for i1 = 1:ip %Set mesh plot dynamic range to pmin
        for i2 = 1:it
            if TxDAFt_db(i1,i2) < pmin,...
               TxDAFt_db(i1,i2) = pmin; end
            if TxDAFp_db(i1,i2) < pmin,...
               TxDAFp_db(i1,i2) = pmin; end
            if RxDAFt_db(i1,i2) < pmin,...
               RxDAFt_db(i1,i2) = pmin; end
            if RxDAFp_db(i1,i2) < pmin,...
               RxDAFp_db(i1,i2) = pmin; end
            if Two_way_DAFt_db(i1,i2) < pmin,...
               Two_way_DAFt_db(i1,i2) = pmin; end
            if Two_way_DAFp_db(i1,i2) < pmin,...
               Two_way_DAFp_db(i1,i2) = pmin; end
```

```
            end
        end

        figure(1),clf %Plot theta component of transmit array
        meshc(Ut,Wt,TxDAFt_db),grid,axis([-1 1 -1 1 pmin,0]),grid
        axis square
        xlabel('U = sin\theta*cos\phi')
        ylabel('W = cos\theta')
        zlabel('Normalized |F_\theta| (dB)')
        view(45,45)
        title('Transmit DSA Pattern')

        figure(2),clf %Plot phi component of transmit array
        mesh(Ut,Wt,TxDAFp_db),grid,axis([-1 1 -1 1 pmin,0]),grid
        axis square
        xlabel('U = sin\theta*cos\phi')
        ylabel('W = cos\theta')
        zlabel('Normalized |F_\phi| (dB)')
        view(45,45)
        title('Transmit DSA Pattern')

        figure(3),clf %Plot theta component of receive array
        meshc(Ur,Wr,RxDAFt_db),grid,axis([-1 1 -1 1 pmin,0]),grid
        axis square
        xlabel('U = sin\theta*cos\phi')
        ylabel('W = cos\theta')
        zlabel('Normalized |F_\theta| (dB)')
        view(45,45)
        title('Receive DSA Pattern')

        figure(4),clf %Plot phi component of receive array
        meshc(Ur,Wr,RxDAFp_db),grid,axis([-1 1 -1 1 pmin,0]),grid
        axis square
        xlabel('U = sin\theta*cos\phi')
        ylabel('W = cos\theta')
        zlabel('Normalized |F_\phi| (dB)')
        view(45,45)
        title('Receive DSA Pattern')

        figure(5),clf %Plot theta component of two-way pattern
        meshc(Ur,Wr,Two_way_DAFt_db),grid,axis([-1 1 -1 1 pmin,0]),grid
        axis square
        xlabel('U = sin\theta*cos\phi')
        ylabel('W = cos\theta')
        zlabel('Normalized |F_\theta| (dB)')
        view(45,45)
        title('Two-way Pattern')

        figure(6),clf %Plot phi component of two-way pattern
        meshc(Ur,Wr,Two_way_DAFp_db),grid,axis([-1 1 -1 1 pmin,0]),grid
        axis square
        xlabel('U = sin\theta*cos\phi')
        ylabel('W = cos\theta')
        zlabel('Normalized |F_\phi| (dB)')
        view(45,45)
        title('Two-way Pattern')
    end

%end case Calculate

%Case calculate two-way gain of XMTR and RCVR arrays
case 'Gain'
```

```
%Get user input for integration intervals for theta and phi
prompt = {'Number of intervals for theta:',...
          'Number of intervals for phi:'};
dlg_title = 'Set number of integration points';
num_lines = 1;
def = {'4','4'};
answer = inputdlg(prompt,dlg_title,num_lines,def);
% set the number of integration intervals (nt points per interval)
ndivt=str2num(answer{1});
ndivp=str2num(answer{2});

%Get parameters
h_eltype = get(findobj(gcf,'Tag','eltype'),'Value');
    if h_eltype == 1, diptype = 'h'; end
    if h_eltype == 2, diptype = 's'; end
    if h_eltype == 3, diptype = 'i'; end
h_dipdir = get(findobj(gcf,'Tag','dipdir'),'Value');
    if h_dipdir == 1, dipdir = 'x'; end
    if h_dipdir == 2, dipdir = 'z'; end
f = (str2num(get(findobj(gcf,'Tag','frequency'),'String')))*1e6;
h = (str2num(get(findobj(gcf,'Tag','height'),'String')));
thetas = (str2num(get(findobj(gcf,'Tag','thetas'),'String')));
phis = (str2num(get(findobj(gcf,'Tag','phis'),'String')));

Nxt = str2num(get(findobj(gcf,'Tag','Nxt'),'String'));
Nzt = str2num(get(findobj(gcf,'Tag','Nzt'),'String'));
dxt = str2num(get(findobj(gcf,'Tag','dxt'),'String'));
dzt = str2num(get(findobj(gcf,'Tag','dzt'),'String'));
Mxt = str2num(get(findobj(gcf,'Tag','Mxt'),'String'));
Mzt = str2num(get(findobj(gcf,'Tag','Mzt'),'String'));
lxt = str2num(get(findobj(gcf,'Tag','lxt'),'String'));
lzt = str2num(get(findobj(gcf,'Tag','lzt'),'String'));

Nxr = str2num(get(findobj(gcf,'Tag','Nxr'),'String'));
Nzr = str2num(get(findobj(gcf,'Tag','Nzr'),'String'));
dxr = str2num(get(findobj(gcf,'Tag','dxr'),'String'));
dzr = str2num(get(findobj(gcf,'Tag','dzr'),'String'));
Mxr = str2num(get(findobj(gcf,'Tag','Mxr'),'String'));
Mzr = str2num(get(findobj(gcf,'Tag','Mzr'),'String'));
lxr = str2num(get(findobj(gcf,'Tag','lxr'),'String'));
lzr = str2num(get(findobj(gcf,'Tag','lzr'),'String'));

txs_xdist = get(findobj(gcf,'Tag','txs_xdist'),'Value');
txs_peddbxval = get(findobj(gcf,'Tag','txs_peddbx'),'Value');
    if txs_peddbxval==1, txs_peddbx=15; end
    if txs_peddbxval==2, txs_peddbx=20; end
    if txs_peddbxval==3, txs_peddbx=25; end
    if txs_peddbxval==4, txs_peddbx=30; end
    if txs_peddbxval==5, txs_peddbx=35; end
    if txs_peddbxval==6, txs_peddbx=40; end
    if txs_peddbxval==7, txs_peddbx=45; end
    if txs_peddbxval==8, txs_peddbx=50; end
txs_nexpx = get(findobj(gcf,'Tag','txs_nexpx'),'Value');

txs_zdist = get(findobj(gcf,'Tag','txs_zdist'),'Value');
txs_peddbzval = get(findobj(gcf,'Tag','txs_peddbz'),'Value');
    if txs_peddbzval==1, txs_peddbz=15; end
    if txs_peddbzval==2, txs_peddbz=20; end
    if txs_peddbzval==3, txs_peddbz=25; end
    if txs_peddbzval==4, txs_peddbz=30; end
    if txs_peddbzval==5, txs_peddbz=35; end
```

```
    if txs_peddbzval==6, txs_peddbz=40; end
    if txs_peddbzval==7, txs_peddbz=45; end
    if txs_peddbzval==8, txs_peddbz=50; end
txs_nexpz = get(findobj(gcf,'Tag','txs_nexpz'),'Value');

tx_xdist = get(findobj(gcf,'Tag','tx_xdist'),'Value');
tx_peddbxval = get(findobj(gcf,'Tag','tx_peddbx'),'Value');
    if tx_peddbxval==1, tx_peddbx=15; end
    if tx_peddbxval==2, tx_peddbx=20; end
    if tx_peddbxval==3, tx_peddbx=25; end
    if tx_peddbxval==4, tx_peddbx=30; end
    if tx_peddbxval==5, tx_peddbx=35; end
    if tx_peddbxval==6, tx_peddbx=40; end
    if tx_peddbxval==7, tx_peddbx=45; end
    if tx_peddbxval==8, tx_peddbx=50; end
tx_nexpx = get(findobj(gcf,'Tag','tx_nexpx'),'Value');

tx_zdist = get(findobj(gcf,'Tag','tx_zdist'),'Value');
tx_peddbzval = get(findobj(gcf,'Tag','tx_peddbz'),'Value');
    if tx_peddbzval==1, tx_peddbz=15; end
    if tx_peddbzval==2, tx_peddbz=20; end
    if tx_peddbzval==3, tx_peddbz=25; end
    if tx_peddbzval==4, tx_peddbz=30; end
    if tx_peddbzval==5, tx_peddbz=35; end
    if tx_peddbzval==6, tx_peddbz=40; end
    if tx_peddbzval==7, tx_peddbz=45; end
    if tx_peddbzval==8, tx_peddbz=50; end
tx_nexpz = get(findobj(gcf,'Tag','tx_nexpz'),'Value');

txs_ampx = getamplitudes(Mxt,txs_xdist,txs_peddbx,txs_nexpx);
txs_ampz = getamplitudes(Mzt,txs_zdist,txs_peddbz,txs_nexpz);
tx_ampx = getamplitudes(Nxt,tx_xdist,tx_peddbx,tx_nexpx);
tx_ampz = getamplitudes(Nzt,tx_zdist,tx_peddbz,tx_nexpz);

rxs_xdist = get(findobj(gcf,'Tag','rxs_xdist'),'Value');
rxs_peddbxval = get(findobj(gcf,'Tag','rxs_peddbx'),'Value');
    if rxs_peddbxval==1, rxs_peddbx=15; end
    if rxs_peddbxval==2, rxs_peddbx=20; end
    if rxs_peddbxval==3, rxs_peddbx=25; end
    if rxs_peddbxval==4, rxs_peddbx=30; end
    if rxs_peddbxval==5, rxs_peddbx=35; end
    if rxs_peddbxval==6, rxs_peddbx=40; end
    if rxs_peddbxval==7, rxs_peddbx=45; end
    if rxs_peddbxval==8, rxs_peddbx=50; end
rxs_nexpx = get(findobj(gcf,'Tag','rxs_nexpx'),'Value');

rxs_zdist = get(findobj(gcf,'Tag','rxs_zdist'),'Value');
rxs_peddbzval = get(findobj(gcf,'Tag','rxs_peddbz'),'Value');
    if rxs_peddbzval==1, rxs_peddbz=15; end
    if rxs_peddbzval==2, rxs_peddbz=20; end
    if rxs_peddbzval==3, rxs_peddbz=25; end
    if rxs_peddbzval==4, rxs_peddbz=30; end
    if rxs_peddbzval==5, rxs_peddbz=35; end
    if rxs_peddbzval==6, rxs_peddbz=40; end
    if rxs_peddbzval==7, rxs_peddbz=45; end
    if rxs_peddbzval==8, rxs_peddbz=50; end
rxs_nexpz = get(findobj(gcf,'Tag','rxs_nexpz'),'Value');

rx_xdist = get(findobj(gcf,'Tag','rx_xdist'),'Value');
rx_peddbxval = get(findobj(gcf,'Tag','rx_peddbx'),'Value');
    if rx_peddbxval==1, rx_peddbx=15; end
    if rx_peddbxval==2, rx_peddbx=20; end
```

```
            if rx_peddbxval==3, rx_peddbx=25; end
            if rx_peddbxval==4, rx_peddbx=30; end
            if rx_peddbxval==5, rx_peddbx=35; end
            if rx_peddbxval==6, rx_peddbx=40; end
            if rx_peddbxval==7, rx_peddbx=45; end
            if rx_peddbxval==8, rx_peddbx=50; end
        rx_nexpx = get(findobj(gcf,'Tag','rx_nexpx'),'Value');

        rx_zdist = get(findobj(gcf,'Tag','rx_zdist'),'Value');
        rx_peddbzval = get(findobj(gcf,'Tag','rx_peddbz'),'Value');
            if rx_peddbzval==1, rx_peddbz=15; end
            if rx_peddbzval==2, rx_peddbz=20; end
            if rx_peddbzval==3, rx_peddbz=25; end
            if rx_peddbzval==4, rx_peddbz=30; end
            if rx_peddbzval==5, rx_peddbz=35; end
            if rx_peddbzval==6, rx_peddbz=40; end
            if rx_peddbzval==7, rx_peddbz=45; end
            if rx_peddbzval==8, rx_peddbz=50; end
        rx_nexpz = get(findobj(gcf,'Tag','rx_nexpz'),'Value');

        rxs_ampx = getamplitudes(Mxr,rxs_xdist,rxs_peddbx,rxs_nexpx);
        rxs_ampz = getamplitudes(Mzr,rxs_zdist,rxs_peddbz,rxs_nexpz);
        rx_ampx = getamplitudes(Nxr,rx_xdist,rx_peddbx,rx_nexpx);
        rx_ampz = getamplitudes(Nzr,rx_zdist,rx_peddbz,rx_nexpz);

        % Compute Gain of Transmit DSA

[txgain,txemax,txprad]=compute_gain('t',f,dxt,Nxt,dzt,Nzt,lxt,Mxt,lzt,Mzt,h,dip
type,dipdir,thetas,phis,txs_ampx,txs_ampz,tx_ampx,tx_ampz,ndivt,ndivp);
        % Compute Gain of Receive DSA

[rxgain,rxemax,rxprad]=compute_gain('r',f,dxr,Nxr,dzr,Nzr,lxr,Mxr,lzr,Mzr,h,dip
type,dipdir,thetas,phis,rxs_ampx,rxs_ampz,rx_ampx,rx_ampz,ndivt,ndivp);

        txgdb=10*log10(txgain);
        disp(['total radiated power, prad = ',num2str(txprad)])
        disp(['max field value of transmit array, V/m = ',num2str(txemax)])
        disp(['transmit gain = ',num2str(txgain),',  in dB = ',num2str(txgdb)])

        rxgdb=10*log10(rxgain);
        disp(['max field value of receive array, V/m = ',num2str(rxemax)])
        disp(['receive gain = ',num2str(txgain),',  in dB = ',num2str(rxgdb)])

        % Compute two-way gain
        tway_gain=txgain*rxgain;
        tway_gaindb=10*log10(tway_gain);

        nl = sprintf('\n');
        msg1 = 'Transmit Array :';
        msg2 = ['Numeric gain = ',num2str(txgain),...
                ', Gain in dB = ',num2str(txgdb),' dB'];
        msg3 = 'Receive Array :';
        msg4 = ['Numeric gain = ',num2str(rxgain),...
                ', Gain in dB = ',num2str(rxgdb),' dB'];
        msg5 = ['Two-way numeric gain = ',num2str(tway_gain),nl,...
                'Two-way Gain in dB = ',num2str(tway_gaindb),' dB'];

        h = msgbox([msg1 nl msg2 nl nl msg3 nl msg4 nl nl msg5],...
            'Gain Results');

     %end case Gain
```

```
     case 'Close'
        h_figs = get(0,'children');
        for fig = h_figs'
            delete(fig);
        end

end %switch

%%%%%%%%%%%%%%%%%%%%%%%%Functions for input validation%%%%%%%%%%%%%%%%%%%%

% validates Frequency
function freq_out = getFreq(freq)

   temp = str2num(freq);
   if (isempty(temp)) | (temp <= 0)
      errordlg('Enter a postive number.', ...
                'Frequency setting', 'error');
      temp = 100; % default set to 100
   end
   freq_out = temp;
% end getFreq

% validates phi starting angle
function o_pstart = getPStart(start)

  temp1 = str2num(start);
  temp2 = str2num(get(findobj(gcf,'Tag','pstop'),'String'));

  if (isempty(temp1)) | temp1 < 0 | temp1 > 180
        errordlg('Enter a Phi Starting angle between 0 and 180 degrees.', ...
                  'Angle Status', 'error');
        temp1 = 90; % default phi start angle
  elseif (start == 'i' | start == 'j')
        errordlg('Enter a Phi Starting angle between 0 and 180 degrees.', ...
                  'Angle Status', 'error');
        temp1 = 0; % default phi start angle
  elseif temp1 > temp2   % phi start greater than phi stop angle
        errordlg('Phi starting angle is greater than ending angle!', ...
                  'Angle Status','error');
        temp1 = 0; % default phi starting angle
  end
  o_pstart = temp1;
% end getPStart

% validates phi ending angle
function o_pstop = getPStop(stop)

  temp1 = str2num(stop);
  temp2 = str2num(get(findobj(gcf,'Tag','pstart'),'String'));

  if (isempty(temp1)) | temp1 < 0 | temp1 > 180
    errordlg('Enter a Phi ending angle between starting angle and 180
degrees.', ...
                  'Angle Status', 'error');
    temp1 = 180;
  elseif temp2 > temp1   % phi start greater than phi stop angle
    errordlg('Phi ending angle is less than starting angle!', ...
                  'Angle Status','error');
    temp1 = 180; % default phi ending angle
  elseif (stop == 'i' | stop == 'j')
    errordlg('Enter a Phi ending angle between 0 and 180 degrees.', ...
                  'Angle Status', 'error');
```

```
    temp1 = 180; % default phi ending angle
  end
  o_pstop = temp1;
% end getPStop

   % validates theta starting angle
function o_tstart = getTStart(start)

  temp1 = str2num(start);
  temp2 = str2num(get(findobj(gcf,'Tag','tstop'),'String'));

  if (isempty(temp1)) | temp1 < 0 | temp1 > 180
     errordlg('Enter a Theta Starting angle between 0 and 180 degrees.', ...
                  'Angle Status', 'error');
        temp1 = 0; % default theta start angle
  elseif (start == 'i' | start == 'j')
     errordlg('Enter a Theta Starting angle between 0 and 180 degrees.', ...
                  'Angle Status', 'error');
     temp1 = 0; % default theta start angle
  elseif temp1 > temp2   % theta start greater than theta stop angle
     errordlg('Theta starting angle is greater than ending angle!', ...
                  'Angle Status','error');
     temp1 = 0; % default theta ending angle
  end
  o_tstart = temp1;
% end getTStart

% validates theta ending angle
function o_tstop = getTStop(stop)

  temp1 = str2num(stop);
  temp2 = str2num(get(findobj(gcf,'Tag','tstart'),'String'));

  if (isempty(temp1)) | (temp1 < 0) | (temp1 > 180)
     errordlg('Enter a Theta ending angle between starting angle and 180
degrees.', ...
                  'Theta Stop Angle Status', 'error');
     temp1 = 180;       % default theta stop angle
  elseif (temp2 > temp1)   % theta stop less than theta start angle
     errordlg('Theta ending angle is less than starting angle!', ...
                  'Theta Stop Angle Status','error');
     temp1 = 180; % default theta stop angle
  elseif (stop == 'i' | stop == 'j')
        errordlg('Enter a Theta Ending angle between 0 and 180 degrees.', ...
                  'Theta Stop Angle Status', 'error');
        temp1 = 180; % default theta ending angle
  end
  o_tstop = temp1;
% end getTStop

% validates Number of Elements in array or subarray
function nel_out = getNEL(el_str)

    temp = str2double(el_str);
   if (isempty(temp)) || (floor(temp) <= 0)
      errordlg('Enter a postive integer.', ...
                'Number of Elements', 'error');
      temp = 2; % default set to 2
   end
   nel_out = floor(temp);
% end getNEL
```

```matlab
% validates Array Element Spacing
function o_del = getSpacing(str)

   temp = str2double(str);
   if (isempty(temp)) || (floor(temp) < 0)
      errordlg('Please check spacing of array elements.',...
               '  Element Spacing', 'error');
      temp = 1; % default
   end
   o_del = temp;
% end getSpacing

% validates Scan Angle for Theta and Phi
function o_scan = getScanAngle(scan_angle)

  temp = scan_angle;
  if (isempty(temp)) | (temp < 0) | (temp > 180)
     errordlg('Enter a Scan Angle between 0 and 180 degrees.', ...
               '  Scan Angle', 'error');
     temp = 90; % default scan angle
  end
  o_scan = temp;
% end getScanAngle
```

```
function [SAFtheta,SAFphi,DAFtheta,DAFphi,U,W]=caf_hdip2(tr,f,dx,Nx,...
          dz,Nz,lx,Mx,lz,Mz,h,dipdir,thetas,phis,tstart,tstop,...
          pstart,pstop,delt,delp,ampxs,ampzs,ampx,ampz)
% caf_hdip2.m
% Version: 2.2
% Author: Cher Hock Hin
% Advisor: Professor David C. Jenn
% Date: 16 August 2012
%
% Function computes array pattern for DSA in phi and theta cut (non-dB).
% Linear array of half-way dipoles with a ground plane.
% z is array axis; ground plane is the xz plane.
% y is normal to ground plane.
%
% Function inputs:
% f = frequency
% dx = Subarray element spacing in x-direction in wavelengths
% Nx = Number of subarray elements in x-direction
% dz = Subarray element spacing in z-direction  in wavelengths
% Nz = Number of subarray elements in z-direction
% lx = Subarray spacing in x-direction  in wavelengths
% Mx = Number of subarrays in x-direction
% lz = Subarray spacing in z-direction in wavelengths
% Mz = Number of subarrays in z-direction
% h = height of element above ground plane in wavelengths
% dipdir = dipole direction ("x"=x-direction,"z"=z-direction)
% thetas = theta scan angle
% phis = phi scan angle
% tstart = start angle for theta
% tstop = stop angle for theta
% pstart = start angle for phi
% pstop = stop angle for phi
% delt = theta step size for pattern calculation
% delp = phi step size for pattern calculation
% ampxs = DSA amplitude distribution for x-plane
% ampzs = DSA amplitude distribution for z-plane
% ampx = Subarray amplitude distribution for x-plane
% ampz = Subarray amplitude distribution for z-plane

% Function outputs:
% SAFtheta = Array pattern of single subarray for theta (complex)
% SAFphi = Array pattern of single subarray for phi (complex)
% DAFtheta = Array pattern of DSA for theta (complex)
% DAFphi = Array pattern of DSA for theta (complex)

warning off
rad=pi/180;
wave=3e8/f; %find wavelength
beta=2*pi/wave; %find beta
bk = 2*pi;

%Set for loop values
it=floor((tstop-tstart)/delt)+1;
ip=floor((pstop-pstart)/delp)+1;

%Open waitbar
if tr == 't', txrx = 'transmit'; end
if tr == 'r', txrx = 'receive'; end
msg = ['Computing ',txrx,' array pattern for ',num2str(it*ip),' angles'];
hwait=waitbar(0,msg);
pause(0.1);
```

```
% determine the phase distribution
us = sin(thetas*rad)*cos(phis*rad);
ws = cos(thetas*rad);
psix = bk*dx*us;
psiz = bk*dz*ws;
psix_s = bk*lx*us;
psiz_s = bk*lz*ws;

% generate exact phase required at each element.
% positive scan corresponds to increasing phase lag with increasing n.
xsix = -(2*[1:Nx] - (Nx + 1))/2*psix;
xminx = min(xsix(1),xsix(Nx));
xsix(1:Nx)=xsix(1:Nx)-xminx;
qphx=xsix;
xsiz = -(2*[1:Nz] - (Nz + 1))/2*psiz;
xminz = min(xsiz(1),xsiz(Nz));
xsiz(1:Nz)=xsiz(1:Nz)-xminz;
qphz=xsiz;

% generate exact phase required at each subarray.
% positive scan corresponds to increasing phase lag with increasing n.
xsix_s = -(2*[1:Mx] - (Mx + 1))/2*psix_s;
xminx_s = min(xsix_s(1),xsix_s(Mx));
xsix_s(1:Mx)=xsix_s(1:Mx)-xminx_s;
qphx_s=xsix_s;
xsiz_s = -(2*[1:Mz] - (Mz + 1))/2*psiz_s;
xminz_s = min(xsiz_s(1),xsiz_s(Mz));
xsiz_s(1:Mz)=xsiz_s(1:Mz)-xminz_s;
qphz_s=xsiz_s;

%Preallocate array size
phi=zeros(ip,it); theta=zeros(ip,it);
U=zeros(ip,it); V=zeros(ip,it); W=zeros(ip,it);
AF=zeros(ip,it); AFs=zeros(ip,it);
SAFtheta=zeros(ip,it); SAFphi=zeros(ip,it);
DAFtheta=zeros(ip,it); DAFphi=zeros(ip,it);
count=0;
% begin Pattern loop
for i1=1:ip %Loop phi
    for i2=1:it %Loop theta
        figure(hwait);
        count=count+1;
        waitbar(count/(ip*it),hwait);
        phi(i1,i2) = pstart + (i1 - 1)*delp;
        phr = phi(i1,i2)*rad;
        theta(i1,i2) = tstart + (i2 - 1)*delt;
        thr = theta(i1,i2)*rad;
        st = sin(thr);      ct = cos(thr);
        cp = cos(phr);      sp = sin(phr);
        u = st*cp;          v = st*sp;          w = ct;
        U(i1,i2) = u;       V(i1,i2) = v;       W(i1,i2)=w;
        sumx = 0;
 %Sum to get array factor of subarray--- begin subarray loop ---
        for n = 1:Nx
            nn = (2*n - (Nx + 1))/2;
            argx = bk*dx*u*nn;
            for m = 1:Nz
                mm = (2*m - (Nz+1))/2;
                argz = bk*dz*w*mm;
                phase = qphx(n) + qphz(m);
                sumx = sumx + ampx(n)*ampz(m)*exp(1i*(phase+argx+argz));
            end
```

```
        end  % end subarray loop -----
        sumx_s = 0;
 %Sum to get array factor of distributed subarrays--- begin dsa loop ---
        for n = 1:Mx
            nn = (2*n - (Mx + 1))/2;
            argx_s = bk*lx*u*nn;
            for m = 1:Mz
                mm = (2*m - (Mz+1))/2;
                argz_s = bk*lz*w*mm;
                phase_s = qphx_s(n) + qphz_s(m);
                sumx_s = sumx_s +
                 ampxs(n)*ampzs(m)*exp(1i*(phase_s+argx_s+argz_s));
            end
        end  % end dsa loop -----

        AF(i1,i2) = sumx; %Array factor for each subarray
        AFs(i1,i2) = sumx_s; %Array factor for dsa
        GF = 2*sin(bk*h*v); %Ground plane factor

        if dipdir=='z' %Colinear dipoles (z directed)
            EFtheta=cos(pi*w/2)./sqrt(1-w.^2+1e-5);
            EFphi=0;
        end

        if dipdir=='x' %Parallel case (x directed)
            EFtheta=cp.*w.*cos(pi*u/2)./(1-u.^2+1e-5);
            EFphi=-sp.*cos(pi*u/2)./(1-u.^2+1e-5);
        end

        %Compute Subarray Pattern
        SAFtheta(i1,i2) =  AF(i1,i2)*GF*EFtheta;
        SAFphi(i1,i2) =  AF(i1,i2)*GF*EFphi;
        %Compute DSA Pattern
        DAFtheta(i1,i2) =  AF(i1,i2)*GF*EFtheta*AFs(i1,i2);
        DAFphi(i1,i2) =  AF(i1,i2)*GF*EFphi*AFs(i1,i2);

    end
end    % end of pattern loop

% close waitbar
close(hwait);

return
```

```
function [SAFtheta,SAFphi,DAFtheta,DAFphi,U,W]=caf_sdip2(tr,f,dx,Nx,...
          dz,Nz,lx,Mx,lz,Mz,h,dipdir,thetas,phis,tstart,tstop,...
          pstart,pstop,delt,delp,ampxs,ampzs,ampx,ampz)
% caf_sdip2.m
% Version: 2.2
% Author: Cher Hock Hin
% Advisor: Professor David C. Jenn
% Date: 16 August 2012
%
% Function computes array pattern for DSA in phi and theta cut (non-dB).
% Linear array of short dipoles with a ground plane.
% z is array axis; ground plane is the xz plane.
% y is normal to ground plane.
%
% Function inputs:
% f = frequency
% dx = Subarray element spacing in x-direction in wavelengths
% Nx = Number of subarray elements in x-direction
% dz = Subarray element spacing in z-direction  in wavelengths
% Nz = Number of subarray elements in z-direction
% lx = Subarray spacing in x-direction  in wavelengths
% Mx = Number of subarrays in x-direction
% lz = Subarray spacing in z-direction in wavelengths
% Mz = Number of subarrays in z-direction
% h = height of element above ground plane in wavelengths
% dipdir = dipole direction ("x"=x-direction,"z"=z-direction)
% thetas = theta scan angle
% phis = phi scan angle
% tstart = start angle for theta
% tstop = stop angle for theta
% pstart = start angle for phi
% pstop = stop angle for phi
% delt = theta step size for pattern calculation
% delp = phi step size for pattern calculation
% ampxs = DSA amplitude distribution for x-plane
% ampzs = DSA amplitude distribution for z-plane
% ampx = Subarray amplitude distribution for x-plane
% ampz = Subarray amplitude distribution for z-plane

% Function outputs:
% SAFtheta = Array pattern of single subarray for theta (complex)
% SAFphi = Array pattern of single subarray for phi (complex)
% DAFtheta = Array pattern of DSA for theta (complex)
% DAFphi = Array pattern of DSA for theta (complex)

warning off
rad=pi/180;
wave=3e8/f; %find wavelength
beta=2*pi/wave; %find beta
bk = 2*pi;

%Set for loop values
it=floor((tstop-tstart)/delt)+1;
ip=floor((pstop-pstart)/delp)+1;

%Open waitbar
if tr == 't', txrx = 'transmit'; end
if tr == 'r', txrx = 'receive'; end
msg = ['Computing ',txrx,' array pattern for ',num2str(it*ip),' angles'];
hwait=waitbar(0,msg);
pause(0.1);
```

```
% determine the phase distribution
us = sin(thetas*rad)*cos(phis*rad);
ws = cos(thetas*rad);
psix = bk*dx*us;
psiz = bk*dz*ws;
psix_s = bk*lx*us;
psiz_s = bk*lz*ws;

% generate exact phase required at each element.
% positive scan corresponds to increasing phase lag with increasing n.
xsix = -(2*[1:Nx] - (Nx + 1))/2*psix;
xminx = min(xsix(1),xsix(Nx));
xsix(1:Nx)=xsix(1:Nx)-xminx;
qphx=xsix;
xsiz = -(2*[1:Nz] - (Nz + 1))/2*psiz;
xminz = min(xsiz(1),xsiz(Nz));
xsiz(1:Nz)=xsiz(1:Nz)-xminz;
qphz=xsiz;

% generate exact phase required at each subarray.
% positive scan corresponds to increasing phase lag with increasing n.
xsix_s = -(2*[1:Mx] - (Mx + 1))/2*psix_s;
xminx_s = min(xsix_s(1),xsix_s(Mx));
xsix_s(1:Mx)=xsix_s(1:Mx)-xminx_s;
qphx_s=xsix_s;
xsiz_s = -(2*[1:Mz] - (Mz + 1))/2*psiz_s;
xminz_s = min(xsiz_s(1),xsiz_s(Mz));
xsiz_s(1:Mz)=xsiz_s(1:Mz)-xminz_s;
qphz_s=xsiz_s;

%Preallocate array size
phi=zeros(ip,it); theta=zeros(ip,it);
U=zeros(ip,it); V=zeros(ip,it); W=zeros(ip,it);
AF=zeros(ip,it); AFs=zeros(ip,it);
SAFtheta=zeros(ip,it); SAFphi=zeros(ip,it);
DAFtheta=zeros(ip,it); DAFphi=zeros(ip,it);
count=0;
% begin Pattern loop
for i1=1:ip %Loop phi
    for i2=1:it %Loop theta
        figure(hwait);
        count=count+1;
        waitbar(count/(ip*it),hwait);
        phi(i1,i2) = pstart + (i1 - 1)*delp;
        phr = phi(i1,i2)*rad;
        theta(i1,i2) = tstart + (i2 - 1)*delt;
        thr = theta(i1,i2)*rad;
        st = sin(thr);      ct = cos(thr);
        cp = cos(phr);      sp = sin(phr);
        u = st*cp;          v = st*sp;          w = ct;
        U(i1,i2) = u;       V(i1,i2) = v;       W(i1,i2)=w;
        sumx = 0;
 %Sum to get array factor of subarray--- begin subarray loop ---
        for n = 1:Nx
            nn = (2*n - (Nx + 1))/2;
            argx = bk*dx*u*nn;
            for m = 1:Nz
                mm = (2*m - (Nz+1))/2;
                argz = bk*dz*w*mm;
                phase = qphx(n) + qphz(m);
                sumx = sumx + ampx(n)*ampz(m)*exp(1i*(phase+argx+argz));
            end
```

```
        end  % end subarray loop -----
        sumx_s = 0;
 %Sum to get array factor of distributed subarrays--- begin dsa loop ---
        for n = 1:Mx
            nn = (2*n - (Mx + 1))/2;
            argx_s = bk*lx*u*nn;
            for m = 1:Mz
                mm = (2*m - (Mz+1))/2;
                argz_s = bk*lz*w*mm;
                phase_s = qphx_s(n) + qphz_s(m);
                sumx_s = sumx_s +
                 ampxs(n)*ampzs(m)*exp(1i*(phase_s+argx_s+argz_s));
            end
        end  % end dsa loop -----

        AF(i1,i2) = sumx; %Array factor for each subarray
        AFs(i1,i2) = sumx_s; %Array factor for dsa
        GF = 2*sin(bk*h*v); %Ground plane factor

        if dipdir=='z' %Colinear dipoles (z directed)
            EFtheta=sqrt(1-w.^2);
            EFphi=0;
        end

        if dipdir=='x' %Parallel case (x directed)
            EFtheta=cp.*w;
            EFphi=-sp;
        end

        %Compute Subarray Pattern
        SAFtheta(i1,i2) =  AF(i1,i2)*GF*EFtheta;
        SAFphi(i1,i2) =  AF(i1,i2)*GF*EFphi;
        %Compute DSA Pattern
        DAFtheta(i1,i2) =  AF(i1,i2)*GF*EFtheta*AFs(i1,i2);
        DAFphi(i1,i2) =  AF(i1,i2)*GF*EFphi*AFs(i1,i2);

    end
end    % end of pattern loop

% close waitbar
close(hwait);

return
```

```
function [SAFtheta,SAFphi,DAFtheta,DAFphi,U,W]=caf_iso2(tr,f,dx,Nx,...
          dz,Nz,lx,Mx,lz,Mz,h,thetas,phis,tstart,tstop,...
          pstart,pstop,delt,delp,ampxs,ampzs,ampx,ampz)
% caf_iso2.m
% Version: 2.2
% Author: Cher Hock Hin
% Advisor: Professor David C. Jenn
% Date: 16 August 2012
%
% Function computes array pattern for DSA in phi and theta cut (non-dB).
% Linear array of isotropic elements with a ground plane.
% z is array axis; ground plane is the xz plane.
% y is normal to ground plane.
%
% Function inputs:
% f = frequency
% dx = Subarray element spacing in x-direction in wavelengths
% Nx = Number of subarray elements in x-direction
% dz = Subarray element spacing in z-direction  in wavelengths
% Nz = Number of subarray elements in z-direction
% lx = Subarray spacing in x-direction  in wavelengths
% Mx = Number of subarrays in x-direction
% lz = Subarray spacing in z-direction in wavelengths
% Mz = Number of subarrays in z-direction
% h = height of element above ground plane in wavelengths
% thetas = theta scan angle
% phis = phi scan angle
% tstart = start angle for theta
% tstop = stop angle for theta
% pstart = start angle for phi
% pstop = stop angle for phi
% delt = theta step size for pattern calculation
% delp = phi step size for pattern calculation
% ampxs = DSA amplitude distribution for x-plane
% ampzs = DSA amplitude distribution for z-plane
% ampx = Subarray amplitude distribution for x-plane
% ampz = Subarray amplitude distribution for z-plane

% Function outputs:
% SAFtheta = Array pattern of single subarray for theta (complex)
% SAFphi = Array pattern of single subarray for phi (complex)
% DAFtheta = Array pattern of DSA for theta (complex)
% DAFphi = Array pattern of DSA for theta (complex)

warning off
rad=pi/180;
wave=3e8/f; %find wavelength
beta=2*pi/wave; %find beta
bk = 2*pi;

%Set for loop values
it=floor((tstop-tstart)/delt)+1;
ip=floor((pstop-pstart)/delp)+1;

%Open waitbar
if tr == 't', txrx = 'transmit'; end
if tr == 'r', txrx = 'receive'; end
msg = ['Computing ',txrx,' array pattern for ',num2str(it*ip),' angles'];
hwait=waitbar(0,msg);
pause(0.1);

% determine the phase distribution
```

```
us = sin(thetas*rad)*cos(phis*rad);
ws = cos(thetas*rad);
psix = bk*dx*us;
psiz = bk*dz*ws;
psix_s = bk*lx*us;
psiz_s = bk*lz*ws;

% generate exact phase required at each element.
% positive scan corresponds to increasing phase lag with increasing n.
xsix = -(2*[1:Nx] - (Nx + 1))/2*psix;
xminx = min(xsix(1),xsix(Nx));
xsix(1:Nx)=xsix(1:Nx)-xminx;
qphx=xsix;
xsiz = -(2*[1:Nz] - (Nz + 1))/2*psiz;
xminz = min(xsiz(1),xsiz(Nz));
xsiz(1:Nz)=xsiz(1:Nz)-xminz;
qphz=xsiz;

% generate exact phase required at each subarray.
% positive scan corresponds to increasing phase lag with increasing n.
xsix_s = -(2*[1:Mx] - (Mx + 1))/2*psix_s;
xminx_s = min(xsix_s(1),xsix_s(Mx));
xsix_s(1:Mx)=xsix_s(1:Mx)-xminx_s;
qphx_s=xsix_s;
xsiz_s = -(2*[1:Mz] - (Mz + 1))/2*psiz_s;
xminz_s = min(xsiz_s(1),xsiz_s(Mz));
xsiz_s(1:Mz)=xsiz_s(1:Mz)-xminz_s;
qphz_s=xsiz_s;

%Preallocate array size
phi=zeros(ip,it); theta=zeros(ip,it);
U=zeros(ip,it); V=zeros(ip,it); W=zeros(ip,it);
AF=zeros(ip,it); AFs=zeros(ip,it);
SAFtheta=zeros(ip,it); SAFphi=zeros(ip,it);
DAFtheta=zeros(ip,it); DAFphi=zeros(ip,it);
count=0;
% begin Pattern loop
for i1=1:ip %Loop phi
    for i2=1:it %Loop theta
        figure(hwait);
        count=count+1;
        waitbar(count/(ip*it),hwait);
        phi(i1,i2) = pstart + (i1 - 1)*delp;
        phr = phi(i1,i2)*rad;
        theta(i1,i2) = tstart + (i2 - 1)*delt;
        thr = theta(i1,i2)*rad;
        st = sin(thr);      ct = cos(thr);
        cp = cos(phr);      sp = sin(phr);
        u = st*cp;          v = st*sp;          w = ct;
        U(i1,i2) = u;       V(i1,i2) = v;       W(i1,i2)=w;
        sumx = 0;
 %Sum to get array factor of subarray--- begin subarray loop ---
        for n = 1:Nx
            nn = (2*n - (Nx + 1))/2;
            argx = bk*dx*u*nn;
            for m = 1:Nz
                mm = (2*m - (Nz+1))/2;
                argz = bk*dz*w*mm;
                phase = qphx(n) + qphz(m);
                sumx = sumx + ampx(n)*ampz(m)*exp(1i*(phase+argx+argz));
            end
        end  % end subarray loop -----
```

```
        sumx_s = 0;
 %Sum to get array factor of distributed subarrays--- begin dsa loop ---
        for n = 1:Mx
            nn = (2*n - (Mx + 1))/2;
            argx_s = bk*lx*u*nn;
            for m = 1:Mz
                mm = (2*m - (Mz+1))/2;
                argz_s = bk*lz*w*mm;
                phase_s = qphx_s(n) + qphz_s(m);
                sumx_s = sumx_s +
                 ampxs(n)*ampzs(m)*exp(1i*(phase_s+argx_s+argz_s));
            end
        end  % end dsa loop -----

        AF(i1,i2) = sumx; %Array factor for each subarray
        AFs(i1,i2) = sumx_s; %Array factor for dsa
        GF = 2*sin(bk*h*v); %Ground plane factor

        %Compute Subarray Pattern
        SAFtheta(i1,i2) =  AF(i1,i2)*GF;
        SAFphi(i1,i2) =  AF(i1,i2)*GF;
        %Compute DSA Pattern
        DAFtheta(i1,i2) =  AF(i1,i2)*GF*AFs(i1,i2);
        DAFphi(i1,i2) =  AF(i1,i2)*GF*AFs(i1,i2);

    end
end    % end of pattern loop

% close waitbar
close(hwait);

return
```

```
function [ampnorm] = getamplitudes(N,dist,pedval,nexp)
% filename: getamplitudes.m
%
% Description: This program calculates the amplitude distribution.
% Author:  Prof. David C. Jenn
% Modified by: Cher Hock Hin
% Date:   4 August 2012

% Required subroutines: tayl.m; cosine.m; bayliss.m

% uniform array excitation coefficients (=1/nel)
if dist == 1
    for i = 1:N
    amp(i) = 1/N;
   end
end

% call subroutine to compute taylor coefficients
% NEL MUST BE EVEN
if dist == 2
   amp(1:N) = tayl(N,pedval,nexp);
end

% call subroutine to compute cosine-on-a-pedestal distribution
if dist == 3
   amp(1:N) = cosine(N,pedval,nexp);
end

% call subroutine to compute bayliss distribution for difference beams
% NEL MUST BE EVEN
if dist == 4
   amp(1:N) = bayliss(N,pedval,nexp);
end

% compute triangular coefficients
if dist == 5
    for i = 1:N
        n = (2*i - (N + 1))/2;
        amp(i) = 1 - abs(2*n/N);
    end
end

% normalize all coefficients to the maximum value
ampmax  = max(amp);
ampnorm = amp/ampmax;
```

```
function amp=bayliss(nel,sll,nbar)
%
% SUBROUTINE TO COMPUTE LINEAR BAYLISS COEFFICIENTS FOR DIFFERENCE
% BEAMS. LINEAR BAYLISS DISTRIBUTION
%
      nbar1=nbar-1;
      for i=1:11
        mu(i)=(i-1)+.5;
      end
      sll25=sll-25;
      z(1)=1.87+sll25*.038;
      z(2)=2.50+sll25*.016;
      z(3)=3.35+sll25*.019;
      z(4)=4.25+sll25*.016;
      a=1.45+sll25*.042;
      for ns=5:nbar
        z(ns)=sqrt(a^2+ns^2);
      end
      sigma=mu(nbar+1)/z(nbar);
      del=2/(nel-1);
     for i=1:nel/2
      rho=del/2+del*(i-1);
      for mms=1:nbar
      bb=1;
      for ns=1:nbar1
        bb=bb*(1-(mu(mms)/(sigma*z(ns)))^2);
      end
      bbb=1;
      for lls=1:nbar
        if lls ~= mms
          bbb=bbb*(1-(mu(mms)/mu(lls))^2);
        end
      end
      bes=(-1)^mms;
      b(mms)=mu(mms)^2/bes*bb/bbb;
      end
      gg=0;
      for lls=1:nbar
        pmu=mu(lls)*pi*rho;
        bes=sin(pmu);
        gg=gg+b(lls)*bes;
      end
      gg=abs(gg);
      amp(nel/2+i)=gg;
      amp(nel/2+1-i)=gg;
      end


function amp=cosine(nel,peddb,nexp)
% cosine on a pedestal distribution
% scale the number of elements to correspond to -1 through +1

ped=10^(-peddb/20.);
for n=1:nel
    xn=(n-1)/(nel-1)*2-1;
    amp(n)=(1.-ped)*abs(cos(xn*pi/2))^nexp+ped;
end
```

```
function amp=tayl(noel,slldb,nbar)
%
%     COMPUTES TAYLOR DISTRIBUTION FOR GIVEN SIDELOBE LEVEL AND NBAR
%
%     AMP=ARRAY OF AMPLITUDES COMPUTED BY SUBROUTINE
%     NOEL=NUMBER OF ARRAY ELEMENTS
%     SLL=SIDELOBE LEVEL IN DB
%     NBAR=NBAR IN TAYLOR DISTRIBUTION (.LE.50)
%
      for i=1:noel
        amp(i)=.5;
      end
      if nbar~=1    % if nbar > 1
      dbamp=20/log(10);
      sll=exp(abs(slldb)/dbamp);
      as=log(sll+sqrt(sll^2-1))/pi;
      as=as^2;
      s=nbar^2/(as+nbar^2-nbar+.25);
      nl1=nbar-1;
      for ii=1:nl1
      a1=ii^2/s;
      f(ii)=1;
        for jj=1:nl1
         f(ii)=f(ii)*(1-a1/(as+jj^2-jj+.25));
        end
        for jj=1:ii
         f(ii)=f(ii)/(1+ii/(nbar-jj));
        end
      end
      m2=noel/2;  % noel assumed positive
    if 2*m2 <= noel
      dum=.5;
      for ii=1:nl1
        dum=dum+f(ii);
      end
      amp(m2+1)=2*dum;
    end
   for ii=1:m2
   k=noel+1-ii;
    for jj=1:nl1
     amp(ii)=amp(ii)+f(jj)*cos(pi*jj*(k-ii)/noel);
    end
   amp(ii)=2*amp(ii);
   amp(k)=amp(ii);
  end
  end
```

```
function
[gain,prad,emax]=compute_gain(tr,f,dx,Nx,dz,Nz,lx,Mx,lz,Mz,h,diptype,dipdir,the
tas,phis,ampxs,ampzs,ampx,ampz,ndivt,ndivp)
% compute_gain.m
% Version: 1.1
% Author: Cher Hock Hin
% Advisor: Professor David C. Jenn
% Date: 11 August 2012
%
% Function computes gain for DSA in x-z plane with y normal.
% Uses 20 points Gaussian quadrature integration.
% Reads integration constants from gausq20.m
%
% Function inputs:
% tr = 't' for XMTR and 'r' for RCVR
% f = frequency
% dx = Subarray element spacing in x-direction in wavelengths
% Nx = Number of subarray elements in x-direction
% dz = Subarray element spacing in z-direction  in wavelengths
% Nz = Number of subarray elements in z-direction
% lx = Subarray spacing in x-direction  in wavelengths
% Mx = Number of subarrays in x-direction
% lz = Subarray spacing in z-direction in wavelengths
% Mz = Number of subarrays in z-direction
% h = height of element above ground plane in wavelengths
% diptype = element type ("s"=short,"h"=half-wave,"i"=isotropic)
% dipdir = dipole direction ("x"=x-direction,"z"=z-direction)
% thetas = theta scan angle
% phis = phi scan angle
% ndivt = number of integration intervals for theta
% ndivp = number of integration intervals for phi
% ampxs = DSA amplitude distribution for x-plane
% ampzs = DSA amplitude distribution for z-plane
% ampx = Subarray amplitude distribution for x-plane
% ampz = Subarray amplitude distribution for z-plane

% Function outputs:
% gain = Numerical gain of DSA
% prad = Total radiated power of DSA
% emax = Maximum field value of DSA

rad=pi/180;
wave=3e8/f; %find wavelength
beta=2*pi/wave; %find beta
bk = 2*pi;

% load data for 20-point Gaussian quadrature integration
load gausq20.m
xt=gausq20(:,1);
at=gausq20(:,2);
nt=length(xt);
Time1=cputime;

% integration interval in theta (degrees)
S1=0*rad;
S2=180*rad;
% integration interval in phi (degrees)
Q1=0*rad;
Q2=180*rad;
% generate integration points in theta and phi
% ndivt and ndivp points will be used but the contribution at
% ndivt+1 and ndivp+1
```

```
ds=(S2-S1)/ndivt;
for i=1:ndivt+1
    SS(i)=(i-1)*ds;
    disp(['i,SS(i)= ',num2str(i),', ',num2str(SS(i))])
end
dq=(Q2-Q1)/ndivp;
for i=1:ndivp+1
    QQ(i)=(i-1)*dq;
    disp(['i,QQ(i)= ',num2str(i),', ',num2str(QQ(i))])
end
% subintervals in phi
nphi=0;
for ii=1:ndivp
    P1=dq/2;
    P2=(QQ(ii+1)+QQ(ii))/2;
    for n=1:nt
        nphi=nphi+1;
        wphi(nphi)=at(n);
        phi(nphi)=P1*xt(n)+P2;
    end
end
% subintervals in theta
ntheta=0;
for ii=1:ndivt
    T1=ds/2;
    T2=(SS(ii+1)+SS(ii))/2;
    for i=1:nt
        ntheta=ntheta+1;
        wtheta(ntheta)=at(i);
        theta(ntheta)=T1*xt(i)+T2;
    end
end

ninteg=ntheta*nphi;
if tr=='t', tr='transmit'; end
if tr=='r', tr='receive'; end
msg=['Computing gain for ',tr,' array',sprintf('\n'),...
      'over ',num2str(ninteg),' integration points'];
hwait=waitbar(0,msg);

% determine the phase distribution
us = sin(thetas*rad)*cos(phis*rad);
ws = cos(thetas*rad);
psix = bk*dx*us;
psiz = bk*dz*ws;

% generate exact phase required at each element.
% positive scan corresponds to increasing phase lag with increasing n.
xsix = -(2*[1:Nx] - (Nx + 1))/2*psix;
xminx = min(xsix(1),xsix(Nx));
xsix(1:Nx)=xsix(1:Nx)-xminx;
qphx=xsix;
xsiz = -(2*[1:Nz] - (Nz + 1))/2*psiz;
xminz = min(xsiz(1),xsiz(Nz));
xsiz(1:Nz)=xsiz(1:Nz)-xminz;
qphz=xsiz;

% generate exact phase required at each subarray.
% positive scan corresponds to increasing phase lag with increasing n.
xsix_s = -(2*[1:Mx] - (Mx + 1))/2*psix;
xminx_s = min(xsix_s(1),xsix_s(Mx));
xsix_s(1:Mx)=xsix_s(1:Mx)-xminx_s;
```

```
qphx_s=xsix_s;
xsiz_s = -(2*[1:Mz] - (Mz + 1))/2*psiz;
xminz_s = min(xsiz_s(1),xsiz_s(Mz));
xsiz_s(1:Mz)=xsiz_s(1:Mz)-xminz_s;
qphz_s=xsiz_s;

% compute field at the integration points
emax=0; sum_pwr=0; ism=0;
% begin Pattern loop
for iphi=1:nphi %Loop phi
    for itheta=1:ntheta %Loop theta
        ism=ism+1;
        waitbar(ism/ninteg,hwait);
        thr = theta(itheta);    phr = phi(iphi);
        st = sin(thr);      ct = cos(thr);
        cp = cos(phr);      sp = sin(phr);
        u = st*cp;          v = st*sp;          w = ct;
        sumx = 0;
%Sum to get array factor of subarray--- begin subarray loop ---
        for n = 1:Nx
            nn = (2*n - (Nx + 1))/2;
            argx = bk*dx*u*nn;
            for m = 1:Nz
                mm = (2*m - (Nz+1))/2;
                argz = bk*dz*w*mm;
                phase = qphx(n) + qphz(m);
                sumx = sumx + ampx(n)*ampz(m)*exp(1i*(phase+argx+argz));
            end
        end  % end subarray loop -----
        sumx_s = 0;
%Sum to get array factor of distributed subarrays--- begin dsa loop ---
        for n = 1:Mx
            nn = (2*n - (Mx + 1))/2;
            argx_s = bk*lx*u*nn;
            for m = 1:Mz
                mm = (2*m - (Mz+1))/2;
                argz_s = bk*lz*w*mm;
                phase_s = qphx_s(n) + qphz_s(m);
                sumx_s = sumx_s +
                 ampxs(n)*ampzs(m)*exp(1i*(phase_s+argx_s+argz_s));
            end
        end  % end dsa loop -----

        GF = 2*sin(bk*h*v); %Ground plane factor

        if diptype=='i' %Compute element factor for isotropic
            EFtheta=1; EFphi=-1;
        end

        if diptype=='h' %Compute element factor for half-wave dipole
            if dipdir=='z' %Colinear dipoles (z directed)
                EFtheta=cos(pi*w/2)./sqrt(1-w.^2+1e-5);
                EFphi=0;
            end
            if dipdir=='x' %Parallel case (x directed)
                EFtheta=cp.*w.*cos(pi*u/2)./(1-u.^2+1e-5);
                EFphi=-sp.*cos(pi*u/2)./(1-u.^2+1e-5);
            end
        end

        if diptype=='s' %Compute element factor for short dipole
            if dipdir=='z' %Colinear dipoles (z directed)
```

```
                EFtheta=sqrt(1-w.^2);
                EFphi=0;
            end
            if dipdir=='x' %Parallel case (x directed)
                EFtheta=cp.*w;
                EFphi=-sp;
            end
        end

        % compute emag
        emagt = sumx*sumx_s*GF*EFtheta;
        emagp = sumx*sumx_s*GF*EFphi;
        emagsq = abs(emagt)^2 + abs(emagp)^2;
        if emagsq > emax, emax=emagsq; end % keep track of emax
        xint = emagsq*st;
        sum_pwr = sum_pwr + wphi(iphi)*wtheta(itheta)*xint;
    end
end    % end of pattern loop

% close waitbar
close(hwait);

prad=T1*P1*sum_pwr; % compute radiated power
gain=4*pi*emax/prad; % compute gain

Time2=cputime-Time1;
disp(['runtime for integration: ',num2str(Time2)])

return
```

```
function varargout = two_way_pattern(varargin)
% TWO_WAY_PATTERN M-file for two_way_pattern.fig
%      TWO_WAY_PATTERN, by itself, creates a new TWO_WAY_PATTERN or raises the
existing
%      singleton*.
%
%      H = TWO_WAY_PATTERN returns the handle to a new TWO_WAY_PATTERN or the
handle to
%      the existing singleton*.
%
%      TWO_WAY_PATTERN('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in TWO_WAY_PATTERN.M with the given input
arguments.
%
%      TWO_WAY_PATTERN('Property','Value',...) creates a new TWO_WAY_PATTERN or
raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before two_way_pattern_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to two_way_pattern_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help two_way_pattern

% Last Modified by GUIDE v2.5 04-Aug-2012 21:11:53

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @two_way_pattern_OpeningFcn, ...
                   'gui_OutputFcn',  @two_way_pattern_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before two_way_pattern is made visible.
function two_way_pattern_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to two_way_pattern (see VARARGIN)

% Choose default command line output for two_way_pattern
handles.output = hObject;

% Update handles structure
```

```
guidata(hObject, handles);

% UIWAIT makes two_way_pattern wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = two_way_pattern_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on selection change in eltype.
function eltype_Callback(hObject, eventdata, handles)
% hObject    handle to eltype (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns eltype contents as
cell array
%        contents{get(hObject,'Value')} returns selected item from eltype

% --- Executes during object creation, after setting all properties.
function eltype_CreateFcn(hObject, eventdata, handles)
% hObject    handle to eltype (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function dzt_Callback(hObject, eventdata, handles)
% hObject    handle to dzt (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of dzt as text
%        str2double(get(hObject,'String')) returns contents of dzt as a double
% --- Executes during object creation, after setting all properties.
function dzt_CreateFcn(hObject, eventdata, handles)
% hObject    handle to dzt (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function dxt_Callback(hObject, eventdata, handles)
% hObject    handle to dxt (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of dxt as text
%        str2double(get(hObject,'String')) returns contents of dxt as a double

% --- Executes during object creation, after setting all properties.
function dxt_CreateFcn(hObject, eventdata, handles)
% hObject    handle to dxt (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function Nzt_Callback(hObject, eventdata, handles)
% hObject    handle to Nzt (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Nzt as text
%        str2double(get(hObject,'String')) returns contents of Nzt as a double

% --- Executes during object creation, after setting all properties.
function Nzt_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Nzt (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function Nxt_Callback(hObject, eventdata, handles)
% hObject    handle to Nxt (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Nxt as text
%        str2double(get(hObject,'String')) returns contents of Nxt as a double

% --- Executes during object creation, after setting all properties.
function Nxt_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Nxt (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function Mxt_Callback(hObject, eventdata, handles)
% hObject    handle to Mxt (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Mxt as text
%        str2double(get(hObject,'String')) returns contents of Mxt as a double

% --- Executes during object creation, after setting all properties.
function Mxt_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Mxt (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function Mzt_Callback(hObject, eventdata, handles)
% hObject    handle to Mzt (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Mzt as text
%        str2double(get(hObject,'String')) returns contents of Mzt as a double

% --- Executes during object creation, after setting all properties.
function Mzt_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Mzt (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function lxt_Callback(hObject, eventdata, handles)
% hObject    handle to lxt (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of lxt as text
%        str2double(get(hObject,'String')) returns contents of lxt as a double

% --- Executes during object creation, after setting all properties.
function lxt_CreateFcn(hObject, eventdata, handles)
% hObject    handle to lxt (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function lzt_Callback(hObject, eventdata, handles)
% hObject    handle to lzt (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of lzt as text
%        str2double(get(hObject,'String')) returns contents of lzt as a double

% --- Executes during object creation, after setting all properties.
function lzt_CreateFcn(hObject, eventdata, handles)
% hObject    handle to lzt (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function Mxr_Callback(hObject, eventdata, handles)
% hObject    handle to Mxr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Mxr as text
%        str2double(get(hObject,'String')) returns contents of Mxr as a double

% --- Executes during object creation, after setting all properties.
function Mxr_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Mxr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function Mzr_Callback(hObject, eventdata, handles)
% hObject    handle to Mzr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Mzr as text
%        str2double(get(hObject,'String')) returns contents of Mzr as a double

% --- Executes during object creation, after setting all properties.
function Mzr_CreateFcn(hObject, eventdata, handles)
```

```
% hObject    handle to Mzr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function lxr_Callback(hObject, eventdata, handles)
% hObject    handle to lxr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of lxr as text
%        str2double(get(hObject,'String')) returns contents of lxr as a double

% --- Executes during object creation, after setting all properties.
function lxr_CreateFcn(hObject, eventdata, handles)
% hObject    handle to lxr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function lzr_Callback(hObject, eventdata, handles)
% hObject    handle to lzr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of lzr as text
%        str2double(get(hObject,'String')) returns contents of lzr as a double

% --- Executes during object creation, after setting all properties.
function lzr_CreateFcn(hObject, eventdata, handles)
% hObject    handle to lzr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function dzr_Callback(hObject, eventdata, handles)
% hObject    handle to dzr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of dzr as text
%        str2double(get(hObject,'String')) returns contents of dzr as a double

% --- Executes during object creation, after setting all properties.
```

```
function dzr_CreateFcn(hObject, eventdata, handles)
% hObject    handle to dzr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function dxr_Callback(hObject, eventdata, handles)
% hObject    handle to dxr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of dxr as text
%        str2double(get(hObject,'String')) returns contents of dxr as a double

% --- Executes during object creation, after setting all properties.
function dxr_CreateFcn(hObject, eventdata, handles)
% hObject    handle to dxr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function Nzr_Callback(hObject, eventdata, handles)
% hObject    handle to Nzr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Nzr as text
%        str2double(get(hObject,'String')) returns contents of Nzr as a double

% --- Executes during object creation, after setting all properties.
function Nzr_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Nzr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function Nxr_Callback(hObject, eventdata, handles)
% hObject    handle to Nxr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Nxr as text
%        str2double(get(hObject,'String')) returns contents of Nxr as a double
```

```
% --- Executes during object creation, after setting all properties.
function Nxr_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Nxr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on slider movement.
function thetas_slider_Callback(hObject, eventdata, handles)
% hObject    handle to thetas_slider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider

% --- Executes on selection change in rx_xdist.
function rx_xdist_Callback(hObject, eventdata, handles)
% hObject    handle to rx_xdist (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns rx_xdist contents as
cell array
%        contents{get(hObject,'Value')} returns selected item from rx_xdist

% --- Executes during object creation, after setting all properties.
function rx_xdist_CreateFcn(hObject, eventdata, handles)
% hObject    handle to rx_xdist (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in rx_peddbx.
function rx_peddbx_Callback(hObject, eventdata, handles)
% hObject    handle to rx_peddbx (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns rx_peddbx contents
as cell array
%        contents{get(hObject,'Value')} returns selected item from rx_peddbx

% --- Executes during object creation, after setting all properties.
function rx_peddbx_CreateFcn(hObject, eventdata, handles)
% hObject    handle to rx_peddbx (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
```

```
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in rx_nexpx.
function rx_nexpx_Callback(hObject, eventdata, handles)
% hObject    handle to rx_nexpx (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns rx_nexpx contents as
cell array
%        contents{get(hObject,'Value')} returns selected item from rx_nexpx

% --- Executes during object creation, after setting all properties.
function rx_nexpx_CreateFcn(hObject, eventdata, handles)
% hObject    handle to rx_nexpx (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in rx_zdist.
function rx_zdist_Callback(hObject, eventdata, handles)
% hObject    handle to rx_zdist (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns rx_zdist contents as
cell array
%        contents{get(hObject,'Value')} returns selected item from rx_zdist

% --- Executes during object creation, after setting all properties.
function rx_zdist_CreateFcn(hObject, eventdata, handles)
% hObject    handle to rx_zdist (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in rx_peddbz.
function rx_peddbz_Callback(hObject, eventdata, handles)
% hObject    handle to rx_peddbz (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns rx_peddbz contents
as cell array
%        contents{get(hObject,'Value')} returns selected item from rx_peddbz
```

```
% --- Executes during object creation, after setting all properties.
function rx_peddbz_CreateFcn(hObject, eventdata, handles)
% hObject    handle to rx_peddbz (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in rx_nexpz.
function rx_nexpz_Callback(hObject, eventdata, handles)
% hObject    handle to rx_nexpz (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns rx_nexpz contents as
cell array
%        contents{get(hObject,'Value')} returns selected item from rx_nexpz

% --- Executes during object creation, after setting all properties.
function rx_nexpz_CreateFcn(hObject, eventdata, handles)
% hObject    handle to rx_nexpz (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] M. I. Skolnik, *Introduction to Radar Systems*, 3rd edition. New York: McGraw-Hill, 2001.

[2] "AN/SPY-1 Radar," Available: http://www.globalsecurity.org/military/systems/ship/systems/an-spy-1.htm [Accessed July 18, 2012].

[3] "Sensors, electronic and information warfare systems," Available: http://www.navy.mil/navydata/policy/vision/vis99/v99-ch3d.html [Accessed July 18, 2012].

[4] R. Hermiller, J. Belyea, and P. Tomlinson, "Distributed array radar," *IEEE Transactions on Aerospace and Electronic Systems*, vol. AES-19, no. 6, pp. 831–839, 1983.

[5] C. Lin, "Distributed subarray antennas for multifunction phased-array radar," Master's thesis, Naval Postgraduate School, Monterey, California, September 2003.

[6] "Microwave devices and radar," class notes for EC4610, Department of Electrical and Computer Engineering, Naval Postgraduate School, Summer 2012.

[7] D. C. Jenn, "Arrays with elements above a ground plane," unpublished.

[8] D. C. Jenn, "Comments on circular polarization," unpublished.

[9] W. L. Stutzman and G. A. Thiele, *Antenna Theory and Design*, 2nd edition. Hoboken, NJ: John Wiley & Sons, 1998.

[10] M. I. Skolnik, *Radar Handbook*, 3rd edition. New York: McGraw-Hill, 2008.

[11] A. P. Goffer, M. Kam and P. R. Herczfeld, "Design of phased arrays in terms of random subarrays," *IEEE Transactions on Antenna and Propagation*, vol. 42, no. 6, pp. 820–826, 1994.

[12] C. T. Lin and H. Ly, "Sidelobe reduction through subarray overlapping for wideband arrays," *Proceedings for 2001 IEEE Radar Conference*, Atlanta, GA, pp. 228–233, 2001.

[13] D. C. Jenn, "Digital antennas," unpublished.

[14] D.C. Jenn et al., "Distributed phased arrays and wireless beamforming networks," *International Journal of Distributed Sensor Networks*, vol. 5, pp. 283–302, 2009.

[15] R. J. Mailloux, *Phased Array Antenna Handbook*. Norwood, MA: Artech House Inc., Norwood, 1994.

[16] C. A. Balanis, *Antenna Theory*. New York: Harper & Row, 1982.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Professor R. Clark Robertson
   Chairman, Department of Electrical & Computer Engineering
   Naval Postgraduate School
   Monterey, California

4. Professor David C. Jenn
   Professor, Department of Electrical & Computer Engineering
   Naval Postgraduate School
   Monterey, California

5. Professor Tri T. Ha
   Professor, Department of Electrical & Computer Engineering
   Naval Postgraduate School
   Monterey, California

6. Professor Yeo Tat Soon
   Director, Temasek Defence Systems Institute (TDSI)
   National University of Singapore
   Singapore

7. Ms. Tan Lai Poh
   Senior Manager, Temasek Defence Systems Institute (TDSI)
   National University of Singapore
   Singapore

8. Mr. Fong Saik Hay
   Chief Technology Officer, ST Engineering
   Singapore

9. Mr. Kum Chee Meng
   Chief Technology Officer, ST Electronics
   Singapore

10. Mr. Teai Yam Koon
Senior Vice-President, ST Electronics (Satcom & Sensor Systems)
Singapore

11. Mr. Cher Hock Hin
Assistant Principal Engineer, ST Electronics (Satcom & Sensor Systems)
Singapore